2019-06

DEVELOPMENT OF A ROCKET TEST PLATFORM CAPABLE OF DELIVERING STANDARD DIMENSION PAYLOADS TO NEAR-SPACE ALTITUDES

Pierce, Dillon T.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DEVELOPMENT OF A ROCKET TEST PLATFORM CAPABLE OF DELIVERING STANDARD DIMENSION PAYLOADS TO NEAR-SPACE ALTITUDES

by

Dillon T. Pierce

June 2019

Thesis Advisor: James H. Newman
Co-Advisor: Christopher G. Smithtro

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE DEVELOPMENT OF A ROCKET TEST PLATFORM CAPABLE OF DELIVERING STANDARD DIMENSION PAYLOADS TO NEAR-SPACE ALTITUDES	**5. FUNDING NUMBERS**
6. AUTHOR(S) Dillon T. Pierce	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Within the NPS Small Satellite Lab, research and development of emerging technologies and their implementation as CubeSat payloads has continued to be a focus of hands-on, lab-based academics. Testing of these student-built CubeSats in real-world environmental conditions began with the high-altitude balloon project in which large Nomex weather balloons were used to carry student-built CubeSat payloads to near-space altitudes. Since the inaugural flight in 2011, the high-altitude balloon project has generated interest from both senior civilian and military leadership, as the academic and military utility of such a program is readily apparent. Building upon the success of the program, this thesis seeks to complement the capabilities of the high-altitude balloon as an academic education tool and military payload delivery vehicle through the development of a high-power rocket program within the NPS Space Systems Academic Group. In addition to providing procedures and documentation as to the requirements for establishing a high-power rocket program, this thesis details the design, construction, and proof-of-concept flight of a reusable rocket capable of delivering standard dimension payloads to near-space altitudes. Results from several rocket flights are then used to explore the academic and military utility of a sustainable high-power rocket program.

14. SUBJECT TERMS amateur high-powered rocketry, National Association of Rocketry, Tripoli Rocketry Association, NPS High-Powered Rocket Program, CubeSat, Small Satellite Laboratory, Space Systems Academic Group	**15. NUMBER OF PAGES** 263
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DEVELOPMENT OF A ROCKET TEST PLATFORM CAPABLE OF DELIVERING STANDARD DIMENSION PAYLOADS TO NEAR-SPACE ALTITUDES

Dillon T. Pierce
Captain, United States Marine Corps
BS, U.S. Naval Academy, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SPACE SYSTEMS OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
June 2019

Approved by: James H. Newman
Advisor

Christopher G. Smithtro
Co-Advisor

James H. Newman
Chair, Department of Space Systems Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Within the NPS Small Satellite Lab, research and development of emerging technologies and their implementation as CubeSat payloads has continued to be a focus of hands-on, lab-based academics. Testing of these student-built CubeSats in real-world environmental conditions began with the high-altitude balloon project in which large Nomex weather balloons were used to carry student-built CubeSat payloads to near-space altitudes. Since the inaugural flight in 2011, the high-altitude balloon project has generated interest from both senior civilian and military leadership, as the academic and military utility of such a program is readily apparent. Building upon the success of the program, this thesis seeks to complement the capabilities of the high-altitude balloon as an academic education tool and military payload delivery vehicle through the development of a high-power rocket program within the NPS Space Systems Academic Group. In addition to providing procedures and documentation as to the requirements for establishing a high-power rocket program, this thesis details the design, construction, and proof-of-concept flight of a reusable rocket capable of delivering standard dimension payloads to near-space altitudes. Results from several rocket flights are then used to explore the academic and military utility of a sustainable high-power rocket program.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION AND BACKGROUND....1
A. NPS SSAG HAB PROJECT....2
1. Education and Military Impacts....2
2. HAB Restrictions....3
3. Conclusions....4
B. NPS SSAG HIGH-POWER ROCKET PROGRAM CONCEPT....4
C. ROCKET PROGRAMS....4
1. NASA Sounding Rocket Program....4
2. Air Force Sounding Rocket Program....10
3. Sounding Rocket Program Conclusions....12
D. AMATEUR HIGH-POWERED ROCKETRY....12
1. Education, Science, and Military Impacts....13
2. Cost....14
3. Summary and Conclusions....14
E. CHAPTER SUMMARY AND CONCLUSIONS....15
F. ORGANIZATION OF THESIS....15

II. NPS SSAG HPRP OPERATIONAL CONSIDERATIONS....17
A. CIVILIAN HIGH-POWER ROCKETRY REGULATIONS....17
1. TRA and NAR Certification Levels....18
2. Local Clubs....19
B. MILITARY RULES AND REGULATIONS FOR CONDUCTING HIGH-POWER ROCKETRY ACTIVITIES....19
1. Applicable NAVSEA Instructions and Policies....19
2. QUAL/CERT Program....20
3. Launch Site Locations....21
C. FAR ROCKET RANGE....23
1. FAR Range Facilities....24
2. Operational Area....25
3. Launch Schedule and Fees....29
D. CHAPTER CONCLUSIONS....29

III. PROJECT BACKGROUND....31
A. LEVEL 1 CERTIFICATION (FLIGHT 1)....31
1. Certification Flight....31
2. Lessons Learned....33
B. LEVEL 2 CERTIFICATION (FLIGHT 2)....34

1. Flight 37
2. Lessons Learned....37
C. LEVEL 3 CERTIFICATION (FLIGHT 3) 38
1. Pre-flight Analysis....43
2. Flight 43
3. Lessons Learned....46
D. FAR OPERATIONAL CHECKOUT LAUNCH (FLIGHT 4)....48
1. Pre-flight Analysis....50
2. Flight 51
3. Lessons Learned....55
E. POST-CHECKOUT LAUNCH CONCLUSIONS....56

IV. NPS SSAG HIGH-POWER ROCKET 59
A. ROCKET CHARACTERISTICS 60
B. PLANNED FLIGHT PROFILE....63
C. STABILITY ANALYSIS 67
D. ROCKET COMPONENTS....69
1. Airframe....69
2. Nose Cone 69
3. Recovery System 73
4. Forward Transition 78
5. Rear Transition 79
6. Fins 81
7. Electronics Bay....85
8. Ejection System 87
9. Electronics 91
E. COST....97
F. CHAPTER CONCLUSIONS....98

V. SUSTAINER TEST FLIGHTS (FLIGHTS 5 AND 6)....99
A. FLIGHT 5....102
B. POST-FLIGHT INVESTIGATION ANALYSIS 105
1. Initial Assumptions 107
2. Supporting Evidence....108
3. Anomaly Investigation Conclusions 114
4. Flight 5 Lessons Learned....115
C. SUSTAINER RE-DESIGN 116
1. Rocket Characteristics....117
2. Flight Profile....119
3. Cost and Complexity....121

4. Pre-flight Analysis....121
D. SUSTAINER REDESIGN TEST FLIGHT (FLIGHT 6)....121
1. Flight....122
2. Recovery....127
3. Data Analysis and Lessons Learned....128
4. Flight 6 Lessons Learned....136
5. Flight 6 Conclusions....137

VI. CONCLUSIONS AND FUTURE WORK....139
A. CONCLUSIONS....139
1. Program Operation....139
2. Program Cost....139
B. FUTURE WORK....140
1. Booster Analysis and Subsequent Flight....140
2. Liquid Rocket Engine....140
3. NPS SSAG HPRP Flight Computer....142
4. Curriculum Incorporation....143
5. Milestone Review Procedures and Details....143

APPENDIX A. MATLAB CODE....145
A. ROCKET FLIGHT SIMULATION ESTIMATE – FULL MISSION....145
B. ROCKET FLIGHT SIMULATION ESTIMATE – SUSTAINER ONLY....155
C. MODIFIED BARROWMAN STABILITY CALCULATIONS....163
D. PARACHUTE DESCENT RATE CALCULATOR....167

APPENDIX B. ROCKET SENSOR BOARD SCHEMATIC AND PCB....171
A. PCB TOP VIEW....172
B. PCB BOTTOM VIEW....173

APPENDIX C. SENSOR BOARD SOFTWARE....175
A. MAIN.PY....175
B. TELEMETRY.PY....178
C. MPL3115A2.PY....181
D. IMU.PY [80]....184
E. LSM9DS1.PY [81]....188
F. ADXL377.PY....190
G. ACCELEROMETER_CALIBRATION.PY....192
H. DATA PROCESSING.PY....194

I. UBLOX.PY [82] 197

APPENDIX D. PRE-LAUNCH CHECK AND PACKING LISTS 219
A. CHECK LIST 220
1. L-72 Hours: Make Preparations 220
2. L-48 Hours: Pack 220
3. L-24 Hours: Travel 220
4. L-24 Hours: Launch Setup 221
5. Launch Day 224
B. PACKING LIST 229
1. Nose Cone 229
2. Upper Airframe 229
3. Electronics Bay 229
4. Main Body Tube 230
5. Motor 230
6. Documentation 230
7. Flight Box 231

LIST OF REFERENCES 233

INITIAL DISTRIBUTION LIST 241

LIST OF FIGURES

Figure 1. Current NASA Sounding Rocket Vehicles. Adapted from [5]. 6

Figure 2. Past and Present Sounding Rocket Launch Sites. Adapted from [6]. 8

Figure 3. Friends of Amateur Rocketry Launch Facilities. Adapted from [38]. 25

Figure 4. Marked 7.5 Series Quadrangle Map of the Proposed Operating Area. Adapted from [39] 27

Figure 5. VFR Sectional Map of Flight Operation Area. Adapted from [41]. 28

Figure 6. Level 1 Certification Rocket Landing and Recovery 32

Figure 7. Shear Pin Example 34

Figure 8. Initial Flight Sensor Package Prototype Design 36

Figure 9. Sensor Prototype and Battery Pack Integration 37

Figure 10. Planned Flight Profile of Level 3 Certification Rocket 40

Figure 11. Fully Assembled Level 3 Certification Rocket 41

Figure 12. Level 3 Certification Rocket Flight Electronics 42

Figure 13. Level 3 Certification Launch 44

Figure 14. Level 3 Certification Rocket Flight KML File Rendering 46

Figure 15. Guide Rail Length vs. Velocity for Level 3 Certification Launch 47

Figure 16. Newman 10-Foot Launch Rail. Source: [53]. 49

Figure 17. Guide Rail Length vs. Velocity for FAR Operational Checkout Launch 50

Figure 18. FAR Operational Checkout Rocket Flight KML File Rendering 52

Figure 19. FAR Operational Checkout Rocket Flight KML File Rendering - Impact of Rocket 53

Figure 20. FAR Operational Checkout Nose Cone Recovery 54

Figure 21. FAR Operational Checkout Airframe Recovery 54

Figure 22. FAR Operational Checkout Airframe Crater 55
Figure 23. Super Loki Dart Launch Vehicle. Source: [55]. 60
Figure 24. Scale Drawing of the NPS Class-3 High-Power Rocket 61
Figure 25. CAD Rendering of Rocket Two-Stage Configuration 61
Figure 26. CAD Rendering of Rocket 1st Stage Booster 62
Figure 27. CAD Rendering of Rocket 2nd Stage Sustainer 63
Figure 28. Planned Flight Profile of Sustainer Launch 65
Figure 29. Planned Flight Profile for Full Mission Flight 66
Figure 30. Rocket Stability Diagram 67
Figure 31. Results of Stability Analysis for (a) the Sustainer and (b) the Combined Sustainer and Booster 69
Figure 32. Plot of Von Kármán Profile 70
Figure 33. Upper Nose Cone Electronics Mounting Assembly (Nose Cone – Mount Assembly. SLDASM) 71
Figure 34. Nose Cone Electronics Mounting Sled (Top View) (Nose Cone – Electronics Sled. SLDPRT) 72
Figure 35. Nose Cone Electronics Mounting Sled (Bottom View) (Nose Cone – Electronics Sled. SLDPRT) 73
Figure 36. Sustainer Recovery Configuration 75
Figure 37. Booster Recovery Configuration 76
Figure 38. Sustainer Recovery Flight Profile 77
Figure 39. CAD Rendering of Forward Transition (Transition – New.SLDPRT) 79
Figure 40. Sectional CAD Rendering of Rear Transition (Transition – Sustainer to Booster.SLDPRT) 80
Figure 41. CAD Rendering of Staging Configuration 81
Figure 42. Sustainer (left) and Booster (right) Fin Dimensions 82
Figure 43. Sustainer Fin Construction 83

Figure 44. Sustainer Fin Epoxy Fillets 84
Figure 45. Sustainer Fin Can with Added Carbon Fiber Layer 84
Figure 46. CAD Rendering of Electronics Bay (Electronics Bay.SLDASM) 85
Figure 47. Cutaway CAD Rendering of Electronics Bay (Electronics Bay.SLDASM) 86
Figure 48. Electronics Mounting Sled (Sustainer E-Bay Sled.SLDPRT) 86
Figure 49. RAPTOR CO2 Ejection System. Adapted from [62]. 87
Figure 50. Filament Style Ejection Canister 88
Figure 51. Payload Ejection Test 90
Figure 52. Flight Electronics Simplified Wiring Diagram 92
Figure 53. Custom Sensor HAT and RPi 3 A+ 95
Figure 54. Sensor Communication Protocol 96
Figure 55. Simplified Software Flow Diagram for Custom Sensor Board 97
Figure 56. Newman Launch Rack at the FAR Rocket Range. Source: [53]. 100
Figure 57. CAD Rendering of Rear Sustainer Guide (Rear Fin Guide.SLDPRT) 101
Figure 58. Loading of Sustainer Test Flight Launch 101
Figure 59. Sustainer Test Flight Initial Flight Path Deviation 102
Figure 60. Sustainer Test Flight - Flight Path 103
Figure 61. BigRedBee GPS Data KML File Rendering 104
Figure 62. Payload TT&C Ground Plane Before and After. Adapted from [77]. 105
Figure 63. Rocket and Payload Debris Field 106
Figure 64. Launch Guide Rail Gap 108
Figure 65. Sustainer Fin Impact Damage 109
Figure 66. Sustainer Test Flight Accelerometer Data 110
Figure 67. Sustainer Test Flight Gyroscope Data 111

Figure 68. Post-flight Video Analysis Anomaly Detection 112
Figure 69. Frame-by-Frame View of Test Flight Anomaly 113
Figure 70. CAD Rendering of Sustainer Redesign 116
Figure 71. Prelaunch Photo of Rocket, Author (middle), Thesis Advisor (left), and SSAG Machinist (right) 117
Figure 72. Sustainer Redesign Components 118
Figure 73. Sustainer Redesign Stability Analysis with (a) 20" (51 cm) Payload Bay and (b) 50" (127 m) Payload Bay 118
Figure 74. Planned Flight Profile of Sustainer Redesign Launch 120
Figure 75. Rocket Stability Confirmation 123
Figure 76. Screen Capture at Apogee of On-board Rocket Video 124
Figure 77. Sustainer Test Flight Barometer Data Altitude vs. Time Graph 125
Figure 78. Sustainer Test Flight Barometer Data Velocity vs. Time Graph 126
Figure 79. Sustainer Test Flight Barometer Data Velocity (End of Flight) vs. Time Graph 127
Figure 80. Annotated Rocket Recovery Photo. Adapted from [78]. 128
Figure 81. Comparison of New and Recovered RPi 129
Figure 82. Sustainer Test Flight IMU Acceleration Data 130
Figure 83. Sustainer Test Flight High-G Accelerometer Data 131
Figure 84. Sustainer Test Flight IMU Gyroscope Data 132
Figure 85. 2D KML Rendering of TeleMega GPS Telemetry Data 133
Figure 86. Motor Retention Device 134
Figure 87. Drogue Parachute Shock Cord Motor Retention Device 135
Figure 88. Screen Capture of Onboard Video Revealing Twisted Shock Cord 136
Figure 89. Flight 6 Dynamic Pressure as a Function of Time and Altitude 141

LIST OF TABLES

Table 1. NASA Vehicle Numbers and Corresponding Sounding Rocket Names. Adapted from [5]. 7

Table 2. Correlation of Launch Site Figure Letter and Name. Adapted from [6]. 8

Table 3. Rocket Motor Impulse Classification Guide. Adapted from [13]. 13

Table 4. FAA Rocket Class Definition. Adapted from [22]. 17

Table 5. High-Power Rocketry Certification Level Definitions. Adapted from [24]. 18

Table 6. High-Power Rocketry Certification Level Requirements. Adapted from [24]. 18

Table 7. FAR Licenses. Adapted from [37]. 24

Table 8. Flight 3 Pre-flight Analysis Summary 43

Table 9. Flight 4 Pre-flight Analysis Summary 50

Table 10. Key Flight Performance Parameters for NPS SSAG High-Power Rocket 64

Table 11. Stability Analysis Results for NPS SSAG High-Power Rocket Single Stage and Full Mission Configurations 68

Table 12. Sectional Parachute and Shock Cord Configurations 74

Table 13. Section Recovery Dimensions 78

Table 14. Section Recovery Descent Rates and Time 78

Table 15. Ejection Charge Sizing 89

Table 16. Rocket Electronic Devices 91

Table 17. Flight Computer Responsibilities and Timing 93

Table 18. Flight 6 Pre-flight Analysis Summary 121

Table 19. Sustainer Test Flight Key Flight Parameters 129

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ABRES	advanced ballistic missile reentry system
AGL	above ground level
AMW	Animal Motor Works
APRS	automatic packet reporting system
BLM	Bureau of Land Management
CAD	computer aided design
CFR	Code of Federal Regulations
CG	center of gravity
CP	center of pressure
CSXT	Civilian Space Exploration Team
CTI	Cesaroni Technologies Incorporated
DoD	Department of Defense
DODIC	Department of Defense Identification Code
EEPROM	electrically erasable programmable read-only memory
FAA	Federal Aviation Administration
FAR	Friends of Amateur Rocketry
GPIO	general purpose input output
GPS	global positioning system
HAB	high-altitude balloon
HAT	hardware attached on top
HPRP	high-power rocket program
ICBM	intercontinental ballistic missile
I^2C	inter-integrated circuit
IMU	inertial measurement unit
ISR	information, surveillance, reconnaissance
KML	keyhole markup language
LED	light emitting diode
MSL	mean sea level
NAR	National Association of Rocketry
NAVSEA	Naval Sea Systems Command

NFPA	National Fire Protection Association
NOSSA	Naval Ordnance Safety and Security Activity
NPS	Naval Postgraduate School
NSRP	NASA Sounding Rocket Program
OPNAV	Office of the Chief of Naval Operations
OPNAVINST	Office of the Chief of Naval Operations Instruction
PCB	printed circuit board
QUAL/CERT	Qualification/Certification
R&D	research and development
RSLP	Rocket Systems Launch Program
RPi	Raspberry Pi
SD	secure digital
SDR	software defined radio
SECNAV	Secretary of the Navy
SmallSat	small satellite
SMC	Space and Missile Command
SPI	serial peripheral interface
SRP	Sounding Rocket Program
SRPO	Sounding Rocket Program Office
SSAG	Space Systems Academic Group
STEM	science, technology, engineering, and math
TRA	Tripoli Rocketry Association
TT&C	telemetry, tracking, and control
VFR	visual flight rules
VHF	very high frequency

ACKNOWLEDGMENTS

First and foremost, I owe a huge debt of gratitude to my advisor, Dr. Jim Newman. Sir, thank you for giving me the latitude to run with my ideas and having confidence in my abilities. Whether during formal class instruction, quick lab discussions, or late-night chats under the stars, I've learned more in these two years under your direction than I ever thought possible. I am honored to call you my mentor, and I look forward to our future work together.

To my co-advisor, Dr. Christopher Smithtro. Sir, thank you for taking the painstaking time to give each launch and this thesis a critical eye. Your inputs shaped this concept into an executable program that will have a lasting impact for future SSAG students.

To the SSAG staff. Your resolute devotion to students is exceptional and made this thesis possible. Alex Savattone, David Rigmaiden, Levi Owen, Dr. Giovanni Minelli, Jim Horning, Dan Sakoda, Ron Phelps, Dr. Wenschel Lan, and Noah Weitz, I cannot thank you guys enough. I can say with certainty that the SSAG team is one of the best I've ever seen. I am lucky to have been part of it for my short time at NPS, and I will remember fondly our times together in the lab, machine shop, and desert.

To my mom and dad. Thank you guys for always supporting me in whatever I do. I am remarkably fortunate to have you as my parents and couldn't have accomplished this project without you.

To my son Hank, who had to deal with his "dada" always working on rockets. This work would not have been possible without you. Thank you for always putting a smile on my face and being the wonderful little man that you are. I love you, buddy.

Finally and most importantly to my wife, Katie. If I have achieved anything significant in my career, it is because of you. I am exceptionally lucky to call you my better half and could not ask for a more supportive partner. Thank you for always having my back and allowing me to pursue my dreams. You and me together, we could do anything.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION AND BACKGROUND

Within the Naval Postgraduate School (NPS) Small Satellite (SmallSat) Lab, research and development of emerging technologies and their implementation as CubeSat payloads has continued to be a focus of hands-on, lab-based academics. Testing of student-built CubeSats in real-world environmental conditions began with the high-altitude balloon (HAB) project, in which large Nomex weather balloons were used to carry student-built CubeSat payloads to near-space altitudes.

Since the inaugural flight in 2011, the NPS HAB project has generated interest from both senior civilian and military leadership for its technical, educational, and military implications. From a technical and engineering perspective, the HAB project enables high-altitude technology demonstration and research as a low-cost platform for accessing the near-space environment. Educationally, the HAB project serves as a hands-on education tool that reinforces key learning concepts through student interaction with physical hardware. Militarily, the HAB project serves as a payload delivery vehicle capable of temporary augmentation of space-based capabilities in the near-space environment. These factors, in conjunction with the relative low-cost and simplicity of the program, have solidified the HAB program's place within the NPS Space Systems Academic Group (SSAG) SmallSat Lab and curriculum.

Though the academic and possible military utility of the HAB program is readily apparent, there are inherent limitations in using a balloon that restrict the payload's operational range and altitude as a near-space payload test vehicle. To complement the capabilities of the HAB project, this thesis seeks to develop a rocket-based test program within the NPS SSAG. This thesis details the design, construction, and proof-of-concept flights of a reusable rocket capable of delivering payloads to near-space altitudes and complementing the HAB capabilities. To facilitate the operation of the rocket, operational procedures and documentation are also captured. Results from several rocket flights are then used to explore the academic and military utility of a sustainable, high-power rocket program within the NPS SSAG.

A. NPS SSAG HAB PROJECT

Near-space HAB flights are currently being conducted at the NPS SSAG SmallSat Lab for education and research. Initiated in 2011, the HAB project is a real-world, near-space test program in which large Nomex weather balloons are used to carry student-built nanosatellites and payloads to near-space altitudes. Today, more that sixteen NPS HAB payloads have flown as a part of intern work, class projects, directed studies, and thesis research. Lessons learned from each flight have been carried forward and integrated with each respective iteration of the HAB program. As a result of its continued development, the once-initial HAB test program continues to mature as a focus platform for project-based, hands-on learning within the NPS SmallSat Lab.

1. Education and Military Impacts

As an education platform, the HAB project affords students with valuable real-world, hands-on skills that are otherwise hard to achieve in an academic environment. The incorporation of physical hardware and real-world operational and logistical concerns enhances educational objectives as students are forced to work through new and dynamic situations. Additionally, exposure to actual sub-system design and payload integration reinforces key educational learning outcomes of the space operations and space engineering curriculums. The result has been more technically-proficient military officers who have a better understanding of the physical interaction of components and subsystems. Such understanding enables critical thinking in the ever-developing and ever-changing technical warfare environment.

In addition to its educational benefits, the HAB project has demonstrated a potential capability as a military payload delivery vehicle. One example of the HAB's utility for demonstrating military-relevant payloads in real-world conditions was with the successful flight and test of the Software-Assisted Very High Frequency (VHF) Information Overhead Relay-CubeSat (SAVIOR-Cube) developed by MAJ Philip Swintek, USA. As described by MAJ Swintek, "the SAVIOR-Cube is a software-defined radio (SDR) payload operating as a VHF relay via a nanosatellite in low Earth orbit" [1]. To demonstrate the effectiveness of his design and validate the concept of a nanosatellite

VHF relay, Swintek conducted a HAB flight. During the flight, Swintek was able to successfully demonstrate the effectiveness of the SAVIOR-Cube, validate certain assumptions, and form new conclusions [1]. Swintek's successful HAB test demonstrates the ability of the HAB project to conduct technical and military research in the near-space environment.

2. HAB Restrictions

The HAB project has undoubtedly proven itself a valuable education tool and potential military technology demonstration and payload delivery platform in an academic environment. However, there are inherent limitations to using a balloon for lift that restrict its operational range. Because a balloon requires the atmosphere to generate a buoyant force, it is limited in altitude by the density of the surrounding air. Though specially manufactured balloons specifically designed to set altitude records have achieved altitudes near 174,000 ft. (53 km), balloons are generally limited to the 100,000–120,000 ft. (30–37 km) region of the atmosphere [2]. In addition, because balloons ascend at a relatively slow rate, their flight path is more susceptible to the prevailing winds. Depending on the strength of the wind, balloons and their associated payloads can drift over large distances. Such drift can quickly force balloons out of the desired operational area and render their payloads ineffective.

Rockets by contrast are not limited in altitude by the density of the surrounding air. Because a rocket's thrust is generated by propellant carried onboard, rockets can achieve higher altitudes than balloons. Such altitudes enable access to additional layers of the earth's atmosphere and have the potential to increase payload loiter time. Additionally, because rockets ascend at a much faster rate than balloons, their flight path is less susceptible to the prevailing winds. Maintaining an intended flight path can reduce rocket drift and enable increased payload time aloft over the desired operational areas. Based on these two factors, a rocket program within the NPS SSAG would serve to complement the capabilities of the HAB project by enabling access to higher altitudes and reducing the potential for ascent phase payload drift.

3. Conclusions

The HAB project has successfully demonstrated that a hands-on education and payload delivery platform is beneficial to academic, technical, and military research within the SSAG. In addition, the HAB provides quick and inexpensive access to the near-space environment that would be otherwise difficult to achieve. That said, given practical operational considerations, a rocket-delivered payload would serve to complement the capabilities of the HAB and serve as an additional focus for hands-on, lab-based academics.

B. NPS SSAG HIGH-POWER ROCKET PROGRAM CONCEPT

The concept for the NPS SSAG High-Power Rocket Program (HPRP) is a relatively low-cost suborbital rocket test program that enables technical and military research in the near-space environment. The program should be able to complement the capabilities of the HAB project by facilitating research at different near-space altitudes. Additionally, the program should maintain a rapid mission timeline conducive to university research. Lastly, the program should maintain standardized payload integration and launch practices that decrease mission ambiguity and increase the potential for mission success. An added bonus of the program is that many, if not all, of the HPRP processes and capabilities are directly transferable to actual space-based processes and capabilities.

C. ROCKET PROGRAMS

To gain an understanding of how the goals of the NPS SSAG HPRP can be achieved, it is useful to examine established suborbital rocket test programs and how they address the previous concepts.

1. NASA Sounding Rocket Program

The NASA Sounding Rocket Program (NSRP) is summarized in NASA's sounding rocket handbook as "a suborbital space flight program that primarily supports NASA-sponsored space and earth sciences research activities, other government

agencies, and international sounding rocket groups and scientists" [3]. Listed by the Sounding Rocket Program Office (SRPO), some of the unique features of sounding rockets include the following:

- Rapid response times
- Long dwell times at apogee
- Direct access to the Earth's mesosphere and lower thermosphere – 130,000 – 400,00 ft. (40 – 120 km)
- Low cost
- Ability to recover and re-fly instruments [4]

Today, with over 2,900 missions flown, the NSRP continues to support its primary mission of earth-science research for university, government, and commercial customers [3]. As a result of the many and varied scientific experiments conducted, the program has provided major contributions across a wide range of earth and space science and systems [3]. The NSRP provides insight into how a suborbital rocket test program can be effectively run, produce meaningful technical results, and extend educational outreach. Many of the standardization practices, operational procedures, and educational concepts can be directly applied to the NPS SSAG HPRP.

a. Mission Variability

The NSRP supports mission variability through the maintenance of 16 different sounding-rocket vehicles that enable technical research between the altitudes of 60 and 870 miles (150 – 1,400 km) [5]. Figure 1 shows the current launch-vehicle variants and depicts the scale of each rocket. Additionally, the NASA vehicle number is listed below each rocket to identify which launch vehicle is depicted. Table 1 correlates the NASA vehicle number below each rocket in Figure 1 to the corresponding launch vehicle variant name. Of note within the image and table is the relatively small number of different rocket designs compared to the total number of variants: there are only five different rocket designs (Black Brant, Oriole, Malemute, Orion, and Lynx) and sixteen variants.

Reuse of the same rocket design with multiple variants facilitates a large range of variability in the rocket's mission profile while decreasing the cost and complexity of the NSRP.

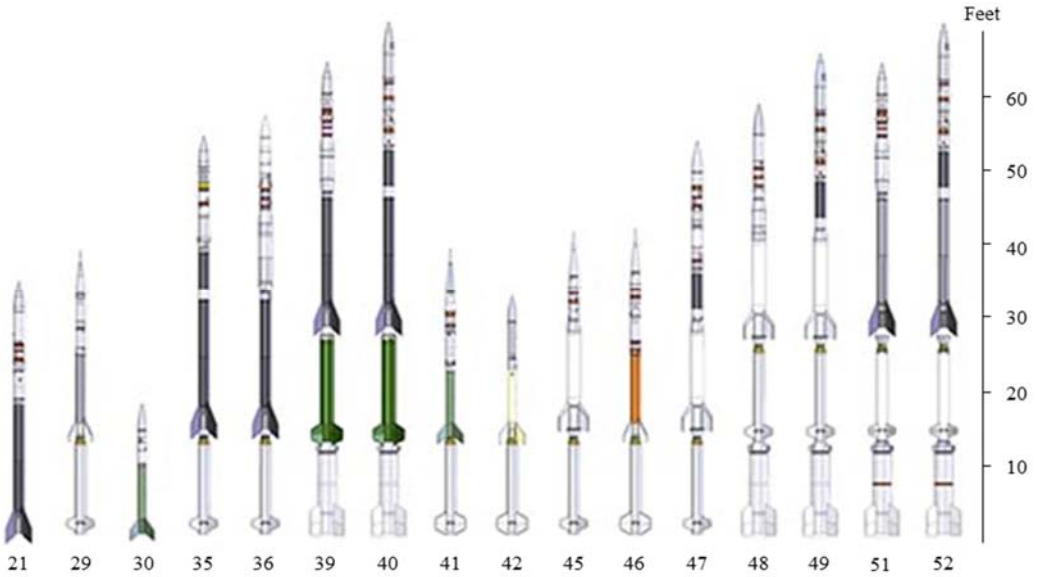


Figure 1. Current NASA Sounding Rocket Vehicles. Adapted from [5].

Table 1. NASA Vehicle Numbers and Corresponding Sounding Rocket Names. Adapted from [5].

NASA Vehicle Number	Sounding Rocket Variant Name
21	Black Brant V
29	Terrier-Malemute
30	Improved Orion
35	Black Brant X
36	Black Brant IX
39	Black Brant XI
40	Black Brant XII
41	Terrier-Improved Orion
42	Terrier-Lynx
45	Oriole II
46	Terrier-Improved Malemute
47	Oriole III-A
48	Oriole III
49	Oriole IV
51	Black Brant XI-A
52	Black Brant XII-A

In addition to maintaining a wide array of launch vehicle variants, the NSRP supports global earth-science research through a relatively large network of launch site locations. Today, there are nine active launch sites that support sounding rocket operations: Wallops Flight Facility, White Sands Test Center, Poker Flats Research Range, Reagan Test Site at Kwajalein Atoll, Andøya Rocket Range, Esrange Space Center, Woomera Rocket Range, Barking Sands Missile Range, and Svalbard Rocket Range [6]. Figure 2 displays the past and present sounding rocket launch locations on a simplified world map. Table 2 correlates each letter in Figure 2 with the respective launch site name. Examination of the table and figure highlights the wide range of sounding rocket operational locations. The wide range of launch facilities enables mission variability and enhances global scientific research.

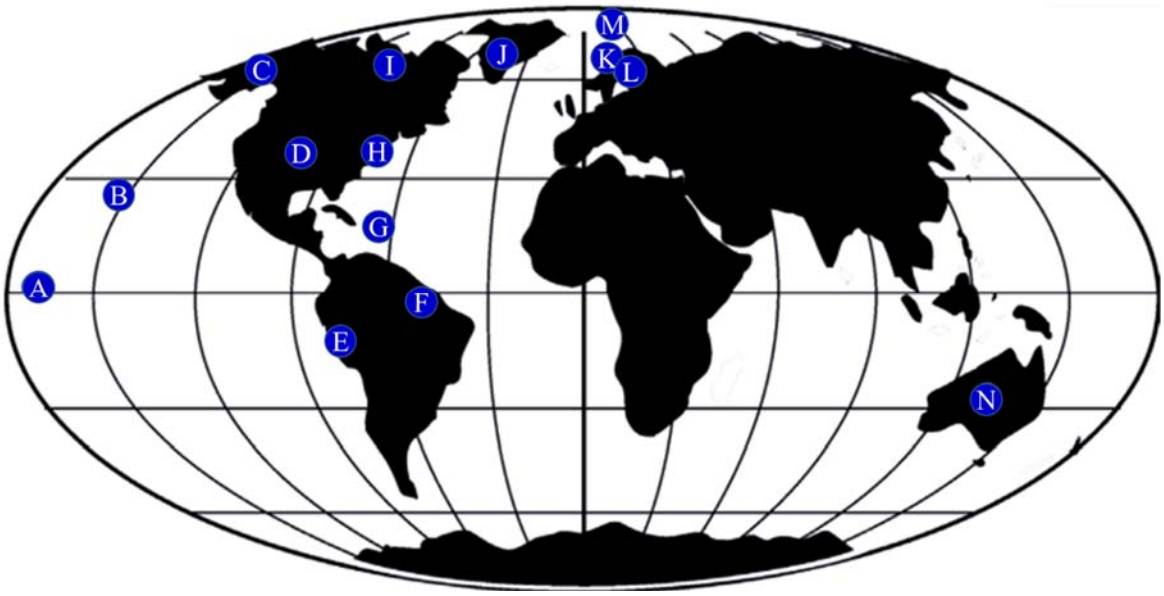


Figure 2. Past and Present Sounding Rocket Launch Sites. Adapted from [6].

Table 2. Correlation of Launch Site Figure Letter and Name. Adapted from [6].

Letter	Launch Site Name
A	Kwajalein Atoll, Marshall Islands
B	Barking Sands, HI
C	Poker Flat, AK
D	White Sands, NM
E	Punta Lobos, Peru *
F	Alcantara, Brazil *
G	Camp Tortuguero, Puerto Rico *
H	Wallops Island, VA
I	Fort Churchill, Canada *
J	Greenland (Thule & Sondre Stromfjord) *
K	Andøya, Norway
L	Esrange, Sweden
M	Svalbard, Norway
N	Woomera, Australia

* Denotes inactive launch sites

b. Affordability and Responsiveness

As previously mentioned, the NSRP maintains program affordability and responsiveness by utilizing a limited number of launch vehicle designs. Operating flight-proven systems both decreases rocket design costs and payload integration ambiguity. In addition, the NSRP maintains programmatic standards and procedures through a consistent mission milestone review process. At a minimum, for each mission, the following milestones must be identified and documented: Launch Date, Launch Time, Integration Site/Date, Mission Initiation Conference, Requirements Definition Meeting, Critical Design Review, Pre-Integration Review, Mission Readiness Review, and Mission Close-out Report [3]. Such processes increase program responsiveness and, according to [3], "facilitate the best possible scientific return from the mission."

The program's low-cost is highlighted by its cost per launch. In 2018, the NASA spending plan allotted $59 million for the sounding rocket program [7]. In that same fiscal year, there were 19 sounding rockets launched as part of the sounding rocket program [6]. Although the cost of each launch varies greatly, given these two numbers, the estimated average cost per launch is approximately $3 million.

c. Educational Impacts

As stated by the SRPO, "sounding rockets provide invaluable tools for education and training" [4]. Research conducted with sounding rockets has been the focus of over 350 Ph.D. dissertations [4]. Additionally, the sounding rocket program maintains community science, technology, engineering, and math (STEM) engagement through six different student opportunity programs at the Wallops Flight Facility: RockSat X, Virginia Space Coast Scholars, RockOn/RockSat-C, Undergraduate Student Instrument Project, and the High Altitude Student Platform [8].

d. NSRP Summary and Conclusions

Since its inception, "the Sounding Rocket Program has provided critical scientific, technical, and educational contributions to the nation's space program" [4]. The program has demonstrated its ability to maintain a low operational cost while facilitating wide

mission variability. Such program characteristics are enabled through a standardization of launch platforms and operational review processes. Over the program's many years of operation, these time-tested systems and processes have proven to both decrease mission lead time and facilitate mission success. Additionally, the program continues to maintain relevance across a wide range of scientific disciplines, enabled by its mission versatility through multiple launch vehicle variants and launch sites. Lastly, the "hands-on" approach to education and training has facilitated STEM outreach and scientific research. In summary, the NSRP, enabled by its rapid response times, technical relevance, and educational impacts is a robust, versatile, and cost-effective flight program that serves as an example for the NPS SSAG HPRP [4].

2. Air Force Sounding Rocket Program

Like the civil and commercial sectors, the military has long been interested in the development and testing of space technologies using suborbital rockets. The Air Force Sounding Rocket Program (SRP) provides an example of military run suborbital rocket test program and its applicability to military research. Many of the same research areas are directly applicable to an NPS SSAG HPRP.

a. Background

In 1963, the military established the Advanced Ballistic Missile Reentry System (ABRES) program to conduct Intercontinental Ballistic Missile (ICBM) research on suborbital rockets [9]. Since its inception, the ABRES program has evolved into what is now known as the Rocket Systems Launch Program (RSLP) [9]. As stated by RSLP officials, the mission of the RSLP is to "provide cost effective space launch and target flight test mission integration and support for national operational and [research and development] requirements on a cost-reimbursable basis" [10]. Under the RSLP, the SRP serves to support this mission as a suborbital test platform for military launch vehicles and payloads and supports both classified and unclassified projects for the Air Force Space Command and Space & Missile Systems Center (SMC) [9].

b. Current Missions and Military Relevance

Currently, the SRP supports a variety of missions, including development of guidance and control systems, missile interceptor target systems and launch vehicle development, development of flight termination systems, and post-launch analysis [9]. Making use of excess ICBM motors, the SRP provides customers with affordable, flexible, and rapid response launches from all over the world [10]. Presently, the program is in its third iteration, known as SRP-3. Documentation of the SRP-3 contract indicates that the SRP is currently involved in the following programs:

- Testing of Patriot missile defense systems with the Juno launch vehicle
- Demonstration of advanced space technologies using Minuteman based systems
- Unspecified Missile Defense Agency tests [11]

Though details of each program are classified, the language in the documentation indicates that any gap in program continuity between the SRP-3 and future contracts would significantly impact these missions and inhibit the program's ability to provide low-cost launch opportunities for its customers [11]. Such language highlights the significance of the program and its potential to facilitate military research and provides potential military research focus areas for the NPS SSAG HPRP.

c. Cost

Details about program costs are difficult to discover in an unclassified environment. However, in 2016, the Air Force Launch Systems Enterprise Directorate held an industry day briefing to discuss the SRP-4 contract. In the briefing, program officials indicated the total price of the contract to be $489 million for approximately 15 anticipated launches [10]. Again, though costs vary greatly with each launch, these two numbers indicate the average cost per launch to be approximately $33 million.

d. Air Force SRP Summary and Conclusions

Though most of the details associated with the Air Force SRP are classified, current unclassified budget figures and mission focus areas highlight the importance of military suborbital rocket testing. Additionally, the SRP's ability to maintain a relatively low operating cost through its re-purposing of excess ICBM motors has enabled the program to succeed and bolstered its mission.

3. Sounding Rocket Program Conclusions

As evidenced by the NSRP and Air Force SRP, suborbital rocket testing has the potential to facilitate technical and military research at a relatively low operational cost, at least on a national scale. However, while the standardization practices, operational procedures, educational concepts, and military research focus areas can be directly applied to the NPS SSAG HPRP, the cost and operational scale of such programs is unrealistic as part of educational classes and research for most educational institutions. Therefore, the construct of the NPS SSAG HPRP must be predicated on a cheaper, less operationally demanding variation of rocket-delivered payload testing. One such variation to the aforementioned professional rocket programs is amateur high-powered rocketry.

D. AMATEUR HIGH-POWERED ROCKETRY

Amateur high-powered rocketry is a hobby similar to model rocketry [12]. The main difference between high-power rockets and model rockets is that high-power rockets use a higher impulse-class (force over time) of rocket motor for propulsion. Though smaller in scale than sounding rockets, they can achieve significant altitudes and would be well suited to serve as the foundation for the NPS SSAG HPRP. Table 3 categorizes the different impulse ranges of rocket motors and delineates the impulse ranges that separate model rockets from high-power rockets.

Table 3. Rocket Motor Impulse Classification Guide. Adapted from [13].

Installed Total Impulse (1bf-seconds) (Newton-seconds)	Motor Type	Classification	Category
0.00 – 36.00 0.00 – 160.00	G or smaller	Model Rocket	Micro to Mid Power
36.01 – 72.00 160.01 – 320.00	H	High Power	Level 1
72.01 – 144.00 320.01 – 640.00	I		
144.01 – 288.00 640.01 – 1,280.00	J		Level 2
288.01 – 576.00 1,280.01 – 2,560.00	K		
576.01 – 1,151.00 2,560.01 – 5,120.00	L		
1,151.01 – 2,302.00 5,120.01 – 10,240.00	M		Level 3
2,302.01 – 4,604.00 10,240.01 – 20,560.00	N		
4,604.01 – 9,208.00 20,560.01 – 40,960.00	O		
9,208.01 – 18,400.00 40,960.01 – 81,920.00	P		Level 3 FAA Permit Req.
18,400.01 – 36,800.00 81,920.01 – 163,840.00	Q		

1. Education, Science, and Military Impacts

As an education tool, high-power rockets provide students with a hands-on platform for the practical implementation of various STEM concepts [14]. Such concepts include physics, calculus, materials and structural engineering, software development and computer programming, electrical engineering, aeronautical engineering, propulsion, and gas dynamics. In addition, students learn real-world practical skills by working through hard integration and operational problems inherent in high-power rocket construction and flight [15].

High-power rockets have the potential for facilitating high-altitude technical and educational research. Though smaller in scale than professional sounding rockets and currently incapable of achieving comparable altitudes, amateur rockets continue to progress in their capabilities. One example of such an achievement occurred on May 17,

2004, when the Civilian Space eXploration Team (CSXT) became the first amateur organization to successfully launch a rocket past the official space boundary of 62 miles (100 km) [16]. Ultimately achieving an estimated altitude of 379,000 ft. (116 km), the CSXT rocket highlighted the potential to conduct small-scale space and near-space research using amateur rocketry [16].

Militarily, amateur rockets can incorporate and test technologies beneficial to the same mission areas that are the focus of the Air Force SRP. Today's amateur rockets, enabled by low-cost microelectronics, are more complex than ever before and are making large strides in rocket technologies previously accessible only to the military and large companies. One example of such advancements is thrust vector control systems that have enabled dynamic stability in amateur rockets [17]. Such a control system incorporates scaled versions of the same hardware and software technologies necessary for large-scale guidance and control systems. Additionally, amateur high-power rockets have the potential to temporarily augment space-based capabilities during times of denial, degradation, and disruption. The rapid launch capability, low cost, and temporary nature of high-power rockets makes their use ideal for localized augmentation of orbital assets in mission areas such as information, surveillance and reconnaissance (ISR), communications, remote sensing, and nuclear command and control assurance.

2. Cost

The cost of amateur high-power rockets can vary widely based on the materials used for construction and the size of motor for propulsion. Construction kits containing all the necessary materials for amateur rocket airframes vary in price from tens of dollars to around a thousand dollars [18]. Additionally, motors that power these rockets range in price from tens of dollars to $10,000 for a Q impulse-class motor [19], [20]. Though these numbers do not capture all possible variations of materials and motors, they do provide some indication of the relatively low cost of amateur high-powered rockets.

3. Summary and Conclusions

Amateur high-powered rocketry as a platform for suborbital payload testing has the potential to provide many of the same technical, military, and educational impacts of

the NSRP and Air Force SRP on a cost and operational scale manageable for a university program. Therefore, amateur high-powered rocketry would be well suited as an initial foundation for the NPS SSAG HPRP.

E. CHAPTER SUMMARY AND CONCLUSIONS

As evidenced by the NSRP and Air Force SRP, suborbital rocket research has the potential to generate significant technical and military outcomes. To achieve comparable research results on a university budget and operational scale, amateur high-power rocketry can be used as platform on which the NPS SSAG HPRP can be based. Within the SSAG, the HAB project has already demonstrated the academic, military, and technical benefit of near-space research. Inclusion of a high-power rocket program in the SSAG as a payload test platform and additional focus for project-based, hands-on learning will expose students to additional educational topics, serve to enhance the space operations and engineering curriculums, and complement the operational capabilities of the HAB project. This thesis establishes the necessary documentation and procedures for the development of such a program as well as produces a functional design and test article for the standard launch vehicle used to baseline the program for the SSAG.

F. ORGANIZATION OF THESIS

Chapter II discusses operational procedures necessary for continuing the establishment and maintaining continuity of the NPS SSAG HPRP. Chapter III provides the program background used to baseline the NPS SSAG high-power rocket launch vehicle design. Chapter IV details the design and construction of a proposed rocket test platform. Chapter V analyzes data from two proof-of-concept flights to generate conclusions and lessons learned for the program and details the design and construction of the program's next-generation rocket. Chapter VI summarizes the author's results and discusses areas of future work for the program.

In addition, Appendix A provides the MATLAB code used to analyze rocket performance and stability characteristics. Appendix B shows the schematic and printed circuit board (PCB) design of the NPS SSAG custom sensor board. Appendix C contains the current version of the PYTHON code used as flight control software for the

developed electronics package. Appendix D is the SSAG rocket flight checklist and packing list.

II. NPS SSAG HPRP OPERATIONAL CONSIDERATIONS

To implement a rocket test program within the NPS SSAG, applicable civilian laws and military regulations that govern high-power rocketry activities must be understood and followed. The purpose of this chapter is to capture the regulations that apply to the NPS SSAG HPRP. Additionally, this chapter discusses various operational considerations in selecting and conducting launches at the program's main launch site.

A. CIVILIAN HIGH-POWER ROCKETRY REGULATIONS

Civilian high-powered rocketry activities are governed by federal, state, and local laws [12]. Federally, the Federal Aviation Administration (FAA) oversees civilian amateur rocketry activities and is responsible for enforcement of amateur rocket regulations dictated in 14 Code of Federal Regulations (CFR) Part 101 Subpart C [21]. Under 14 CFR § 101.22, classes of amateur rockets are delineated by parameters categorized in Table 4.

Table 4. FAA Rocket Class Definition. Adapted from [22].

Class	Definition
Class 1 – Model Rocket	- Uses no more than 4.4 ounces (125 grams) of propellant - Uses a slow-burning propellant - Is made of paper, wood, or breakable plastic - Contains no substantial metal parts - Weighs no more than 53 ounces (1,500 grams) , including the propellant
Class 2 – High-Power Rocket	- Amateur rocket other than model rocket - Contains combined motor impulse of 9,208 lbf-s (40,960 N-s) or less
Class 3 – Advanced High-Power Rocket	- Amateur rocket other than model or high-power rocket (e.g., total installed impulse > 9,208 lb.-s (40,960 N-s))

Locally, it is in the purview of each authority having jurisdiction to adopt codes and standards as they apply to high-power rocketry [23]. To promote standardization, many states and counties have adopted the National Fire Protection Association (NFPA) Code 1127 to establish safe practices for high-power rocketry operations [12]. Within the NFPA Code 1127, two civilian organizations are recognized as national certifying

associations of both high-power rocket motors and users: the National Association of Rocketry (NAR) and the Tripoli Rocketry Association (TRA) [13]. Both organizations promote safe flight practices of high-power rockets through the promulgation of rocketry education and the regulatory oversight of high-power rocket activities [12].

1. TRA and NAR Certification Levels

Within each national association, there are three levels of high-power-rocketry certification that enable individuals to purchase and fly successively larger rocket motors: Level 1, Level 2, and Level 3. Table 5 lists the various certification levels and correlates the respective certification definition. Table 6 lists the requirements for obtaining each respective certification level.

Table 5. High-Power Rocketry Certification Level Definitions. Adapted from [24].

Certification Level	Definition
Level 1	Allows for purchase and use of high-power rockets with a total installed impulse up to 144 lbf-s (640 N-s)
Level 2	Allows for purchase and use of high-power rockets with a total installed impulse up to 1,151 lbf-s (5,120 N-s)
Level 3	Allows for purchase and use of high-power and advanced high-power rockets with a total installed impulse greater than 1,151 lbf-s (5,120 N-s)

Table 6. High-Power Rocketry Certification Level Requirements. Adapted from [24].

Certification Level	Certification Requirements[1]
Level 1	- Must build, launch, and successfully recover a rocket using a motor in the H-I high-power rocket impulse range
Level 2	- Must possess current Level 1 certification - Must pass Level 2 written exam - Must build, launch, and successfully recover a rocket using a motor in the J-L high-power rocket impulse range
Level 3	- Must possess current Level 2 certification - Must successfully demonstrate electronically-initiated recovery system - Must document design, build, and operational phases of certification attempt - Must have a minimum of two national certification board members approve documentation prior to flight attempt - Must build, launch, and successfully recover a rocket using a motor in the M-O high-power rocket impulse range

[1] Further level-specific certification details are available on each respective association webpage

2. Local Clubs

Local clubs are the most practical way for amateurs to fly high-power rockets as they are sanctioned by the national certifying associations and provide oversight to association members. In addition, local clubs provide insurance protection to association members and hold the necessary waivers from local and federal authorities to legally conduct high-power rocket operations [25]. Though it is not necessary to be certified by the civilian associations to operate high-power rockets, it is necessary to be certified by the civilian associations to conduct rocketry operations at the local club launch sites. Therefore, certification is the most practical way for individuals and small-scale organizations to purchase and fly high-power rockets.

B. MILITARY RULES AND REGULATIONS FOR CONDUCTING HIGH-POWER ROCKETRY ACTIVITIES

Because the NPS SSAG HPRP will operate under the jurisdiction of the Navy, all applicable Navy regulations that apply to high-powered rocketry must be followed. Currently, there are no military orders that specifically address amateur high-power rocketry operations. However, as defined by Department of Defense (DoD) Manual 6055.09, amateur solid rocket propellant meets the definition of Hazard Class 1, Divisions 3 or 4, Group C explosive material (HD 1.3/4C) [26]. As such, the storage, transportation, and handling of high-power rocket propellant must be in accordance with DoD and service specific explosive safety policies. Within the Navy, the Naval Ordnance Safety and Security Activity (NOSSA), a field activity within the Naval Sea Systems Command (NAVSEA), is the technical authority responsible for providing ordnance safety policy and oversight to the naval enterprise [27].

1. Applicable NAVSEA Instructions and Policies

Produced by NOSSA, the NAVSEA Ordnance Pamphlet 5 contains the minimum criteria for handling, storing, and transporting explosives [28]. In addition, the Chief of Naval Operations Instruction (OPNAVINST) 8023.24C provides directives and guidance for the administration of the Naval Personnel Ammunition and Explosive Handing Qualification and Certification (QUAL/CERT) Program [29].

2. QUAL/CERT Program

The intent of the Navy QUAL/CERT program is to provide individual commands with the programs and procedures necessary for establishing and maintaining a successful explosive safety program [29]. As part of the program, personnel must be trained, found qualified, and certified as explosive handlers prior to manufacturing, handling, transporting, storing, or assembling explosives [29]. As such, NPS personnel engaged in high-power rocket operations must meet the education and certification requirements of the command's QUAL/CERT program prior to handling high-power rocket motors.

Under the direction of OPNAVINST 8023.24C, installation commanding officers are responsible for the establishment and maintenance of the command's explosives handling QUAL/CERT program [29]. As part of the commanding officer's program, he or she may designate a QUAL/CERT Board Chair to perform daily oversight of the program and conduct the required administrative duties [29]. At NPS, Mr. Steven Schnur is the command appointed Explosives Safety Officer and QUAL/CERT Board Chair [30]. He is responsible for the certification of the command's explosive handlers and for the program's administrative oversight [30]. Established under the NPS QUAL/CERT program, the following requirements must be met to be certified as a command certified explosive handler:

1. Enrollment in the Navy medical surveillance program through the filling out of the following forms and completion of the following events:
    - SECNAV 5100/1 "Supervisor's Medical Surveillance and Certification Exam Referral" form
    - OPNAV 8020/6 "Department of the Navy Medical Examiner's Certificate" form
    - Enrollment is the Navy Enterprise and Management System
    - Urinalysis
    - Explosive Handler Physical Screening

2. Completion of the following Navy e-learning courses:
   - AMMO-49
   - Military Munitions Rule Awareness Training Course (NOSSA-MMRAT-2.0)
   - Navy Materials Possibly Presenting an Explosives Hazard (MPPEH) Requirements Training Course (NOSSA-MPPEH-4.0)
3. Reading of the following documents (applicable to SSAG members):
   - NAVSEA OP 5 Chapter 4 “Fire Protection”
   - NAVSEA OP 5 Chapter 13–15 “MPPEH”
   - Standard Operating Procedure (SOP) NPS-SE-SE3202/3203
   - NPS Explosives Safety Required Reading Power Point
   - Propellant Safety Data Sheets
   - Friends of Amateur Rocketry Risk Hazard Analysis
4. Successful board-verified motor assembly proficiency demonstration [30]

These documents are available and maintained on the NPS Explosives Safety SharePoint Site. Access to the site can be coordinated through Mr. Steven Schnur.

3. Launch Site Locations

Within the military there are several ranges that support large research and development (R&D) rocketry operations. Such sites include: White Sands Missile Range, Dugway Proving Ground, and Naval Air Weapons Station China Lake [31], [32], [33]. These ranges are unique from smaller military installations in that their large span control of land and airspace makes them ideal for flights of new, untested systems [31]. In addition, due to the experimental nature of most munitions undergoing test, the facilities have already established experimental ammunition handling procedures that are conducive to conducting tests with explosive material not explicitly listed under a Joint Munitions Command Department of Defense Identification Code (DODIC) [34]. As amateur high-power rocket motors fall into this category, their storage, transportation,

and experimental use is made easier by utilizing these ranges. However, there are prohibitive factors for smaller units in using large military R&D ranges that make them impractical for smaller routine launches.

One such restriction is that most of the large-scale R&D test ranges receive little institutional funding and operate as "reimbursable mission" organizations [35]. As such, the requesting unit or organization must provide funding for facility, range, and operational support of these installations. Depending on the range, facilities, and type of support requested, the associated cost quickly becomes impractical for a university program. Additionally, these sites serve as the primary test ranges for new and unproven military munitions. Therefore, range scheduling congestion can fluctuate drastically and at unexpected times. Both the potential for high cost and having to compete with high priority units for range time make these large ranges impractical as the main launch site for the SSAG HPRP.

An alternative option to launching at large military R&D rocket ranges is to use smaller, closer military ranges such as Camp Roberts and Ft. Hunter Liggett. These ranges appear to be a good solution to accommodating the cumbersome operational restrictions associated with high-power rocketry as they control relatively large areas of land, generally to not cost to use, and are not as busy as larger ranges. However, upon coordination with range officials, the use of these ranges proves problematic and makes their use impractical as the main launch site for the SSAG HPRP for the following reasons:

1. There are no standing procedures for conducting amateur rocketry operations. Therefore, special exemptions to existing range regulations must be conducted for every rocket flight. Such exemptions require the approval from the base commanding officer [30].

2. These smaller ranges are in areas of high-traffic airspace. Therefore, reserving large corridors of airspace for high-power rocketry operations takes advanced coordination with the FAA 45 days prior to the event [21].

3. Smaller ranges do not retain the necessary equipment to support rocketry operations. All launch hardware such as launch towers, ignition boxes, test stands, etc., must be brought by the user for use.

4. Smaller installations do not retain standing exemptions for storage and use of non-DODIC munitions. Therefore, they must request special authorization for high-power rocketry operations.

Due to these restrictions, military ranges are an impractical solution for the low cost and short lead-time scheduling objectives of the NPS SSAG HPRP. Therefore, civilian experimental rocketry sites are explored as an alternative. The Friends of Amateur Rocketry (FAR) Range in Mojave, CA provides the solution for the main launch site for the NPS SSAG HPRP.

C. FAR ROCKET RANGE

The FAR Rocket Range is a non-profit experimental rocket test facility located in Mojave, CA [36]. As a civilian-run rocket range, the FAR Rocket Range retains the necessary federal and local licenses for rocketry operations. Table 7 lists the current federal and local licenses necesseary for high-power rocket operations that are retained by the FAR Rocket Range.

Table 7. FAR Licenses. Adapted from [37].

Organization	FAR License
FAA	Class 2 Rockets - wavier to 50,000 feet (15 km) mean sea level (MSL) Saturday and Sunday and 18,000 feet (5.5 km) MSL Monday through Friday, sunrise to sunset Class 3 Rockets - wavier to 22,000 feet (6.7 km) above ground level (AGL) Saturday and Sunday and 18,000 feet (5.5 km) MSL Monday through Friday, sunrise to sunset
Bureau of Land Management (BLM)	Right-of-way access to FAR property
Kern County Fire Department	Explosives Magazine Permit Rocket Launch Permit
Kern County Health Services Department	Environmental Health Permit
Bureau of Alcohol, Tobacco, Firearms, and Explosives	Explosive Manufacturing Permit
Federal Communications Commission	General Mobile Radio Service
California State Fire Marshal	Class 1, 2, and 3 Pyrotechnic Operator Licenses

1. FAR Range Facilities

In addition to having all the requisite equipment for launch support such as launch racks, towers, and control boxes, the FAR Rocket Range provides several facilities for range users. Figure 3 is an annotated Google Earth satellite image of the FAR Rocket Range and its facilities. Of note, the figure depicts the various support facilities and their respective location on the FAR complex. The support facilities include launch bunkers with reinforced overhead cover, indoor and covered work facilities, a small machine shop, static test firing stands, and propellant storage. In addition, since this image was taken in September of 2015 [38], FAR staff have added an indoor classroom and launch viewing area.

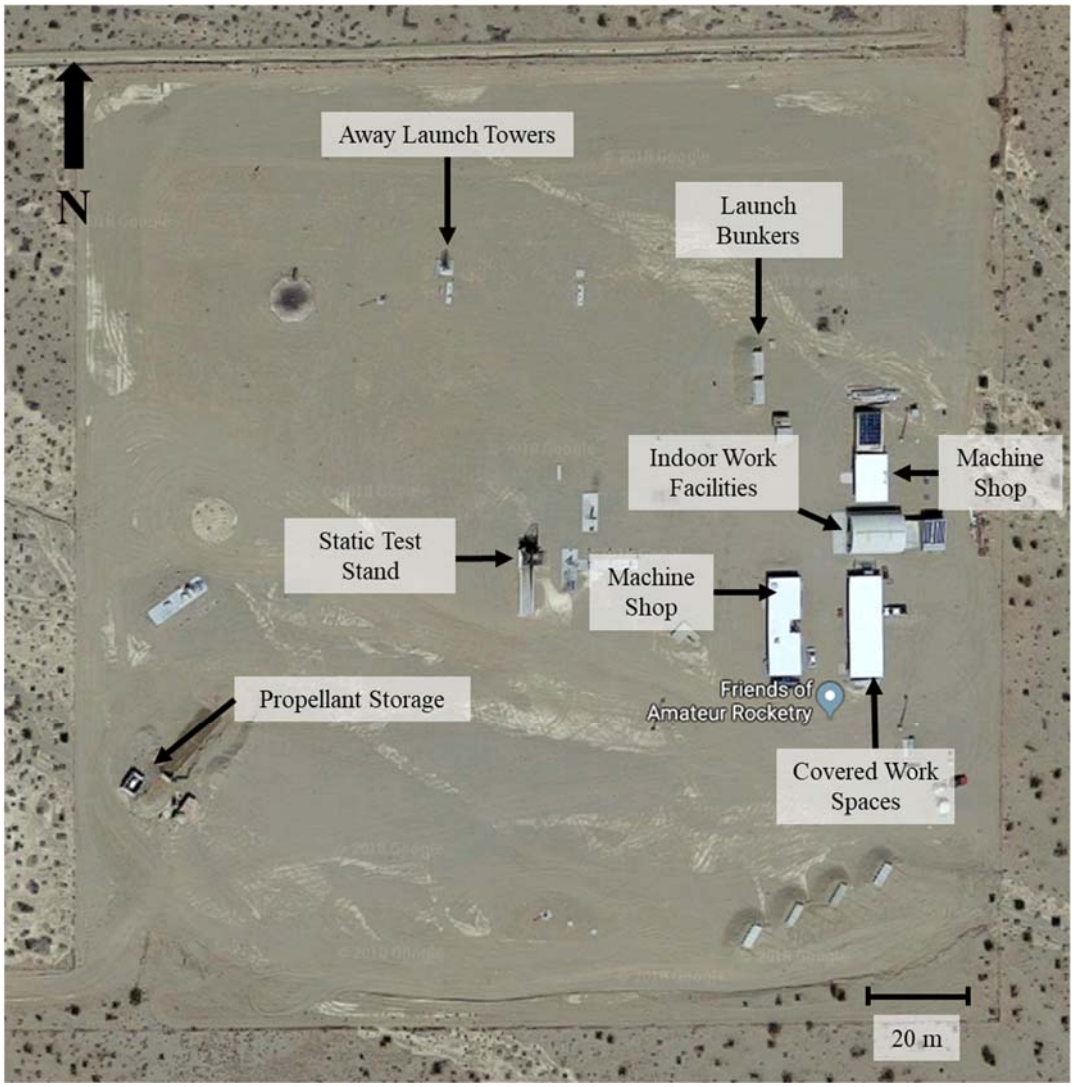


Figure 3. Friends of Amateur Rocketry Launch Facilities. Adapted from [38].

2. Operational Area

The FAR complex itself is relatively small as it encompasses only approximately 11 acres.[1] However, much of the surrounding area is public land managed by the BLM. Such a setup affords a large, relatively unpopulated area around the FAR Range that makes it suitable for experimental rocketry operations. Figure 4 is a marked 7.5 series topographic quadrangle map, published by the U.S. Geological Survey (scale 1:24,000),

[1] Determined via Google Maps distance measurements of the FAR complex.

of the proposed operating area. The FAR launch site is located in the middle of the map with six- and twelve-mile radius circles surrounding the main launch tower. The different color shapes highlight the various terrain and operational restrictions that drive the rocket flight profile determination for this launch site. The image highlights the large areas of potential landing sites around the FAR launch facility.

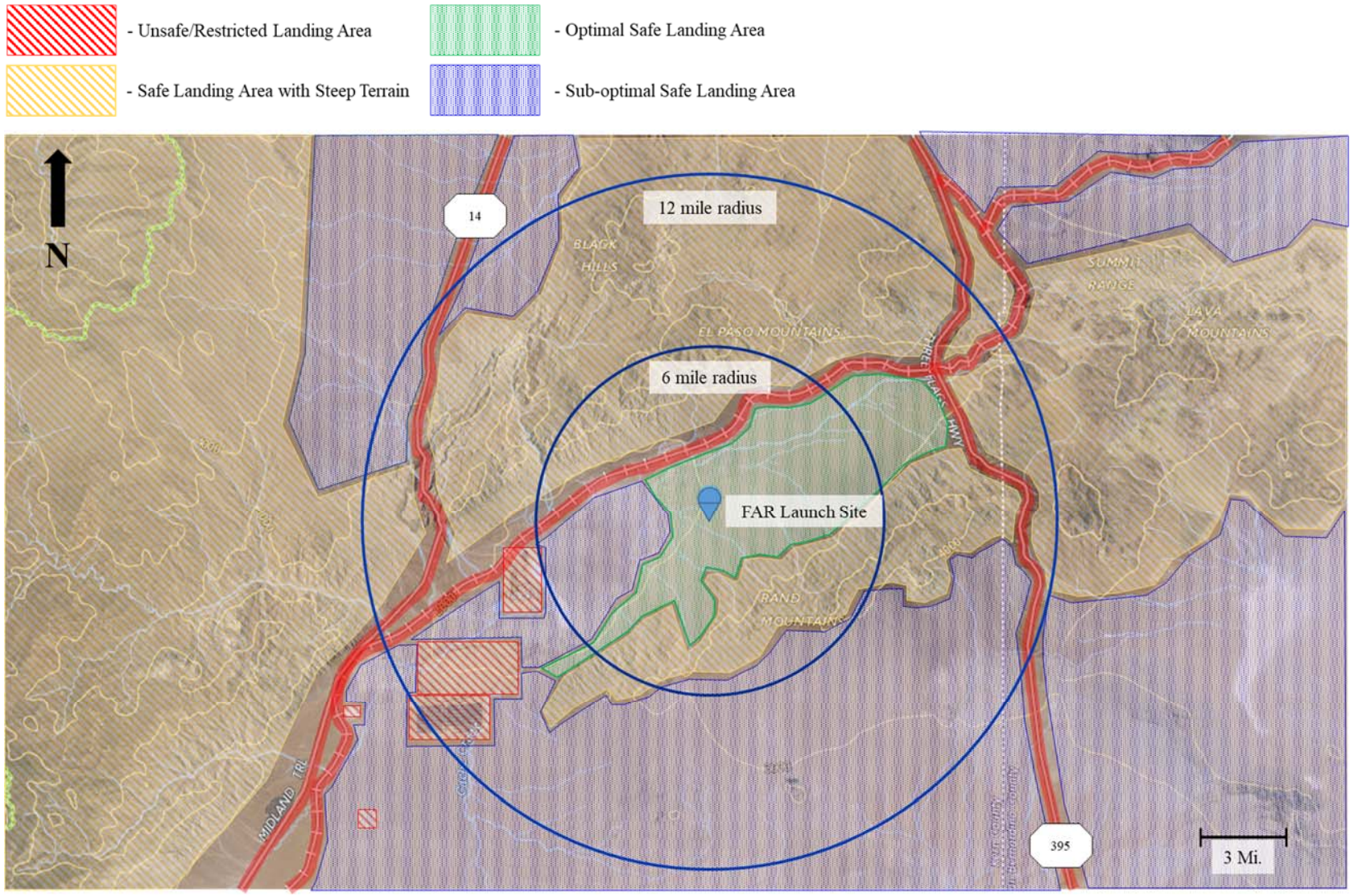


Figure 4. Marked 7.5 Series Quadrangle Map of the Proposed Operating Area. Adapted from [39].

In addition to the ground operational considerations for the FAR launch site, airspace constraints can be visualized via a visual flight rules (VFR) sectional map produced by SkyVector.com shown in Figure 5. The figure indicates that above 3,000 ft. (900 m) AGL, the airspace above the launch site falls within the Isabella military operational area (outlined in red on the map). Additionally, the figure indicates that the launch site is approximately 36,000 ft. (11 km) from the nearest restricted airspace which meets the operational restriction range of 30,400 ft. (9.26 km) of any airport boundary as prescribed in 14 CFR Part 101.25 [40].

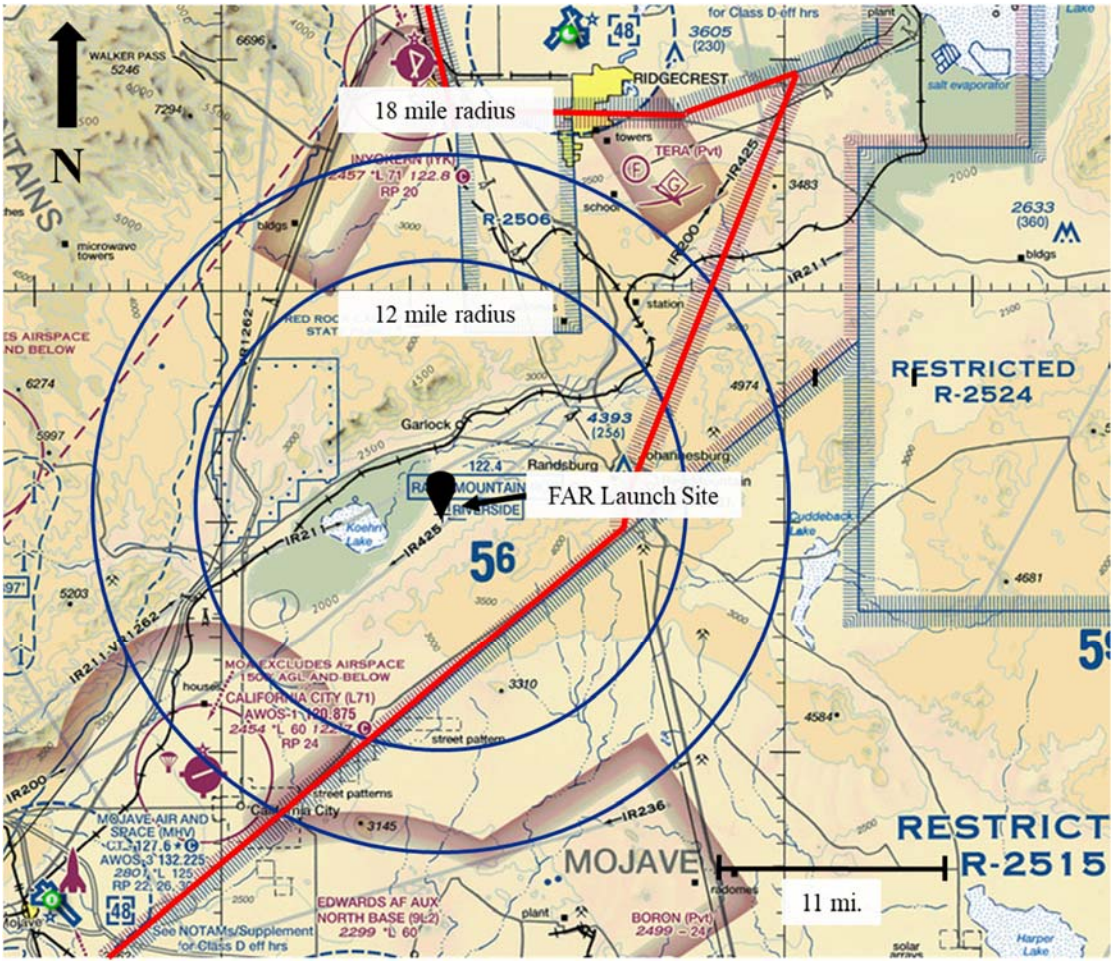


Figure 5. VFR Sectional Map of Flight Operation Area. Adapted from [41].

3. Launch Schedule and Fees

The mission of the FAR Rocket Range is "to educate the general public in STEM through the use of amateur rocketry; and to foster rocket technology by supporting individuals, hobbyists, student groups, businesses, and other likeminded non-profit entities" [42]. In efforts to facilitate this mission, FAR officers have implemented a flexible launch schedule and a relatively low facility use cost.

Every other weekend FAR holds a FAR Launch Day where the facility is open to use by paying individuals and groups [43]. Cost for using the range on a FAR Launch Day is $10 for non-member individual users [44]. In addition, the range is available for private use during both non-FAR Launch Day weekends and weekdays [44]. Cost for renting the range for private use varies on the day, but ranges between $500 and $1,000 [44]. All costs are negotiable, and the staff is very willing to accommodate special circumstances [44].

D. CHAPTER CONCLUSIONS

For the NPS SSAG HPRP to be functional, it must operate within the parameters of the NPS Explosives QUAL/CERT program. Maintenance of qualified and current explosive handlers on the SSAG staff will facilitate high-power rocket operations and expedite student research. In addition, getting SSAG staff members qualified as explosive handlers will increase the continuity of the program's explosive safety knowledge while decreasing the burden on the students conducting high-power rocket operations and research. Based on the explosive hazard regulations, a minimum two SSAG staff members or students will need to be qualified for the program to operate.

The FAR Rocket Range has many favorable attributes that make it suitable for use as the NPS SSAG HPRP main launch site. Primarily, the FAR's relatively low-cost range use fee and flexible schedule decreases the operational cost and complexity of the NPS SSAG HPRP. Additionally, the large expanse of unpopulated land surrounding the launch site make it apt for experimental research and testing. Lastly, the on-site facilities maintained by the FAR decreases the logistical burden on the NPS SSAG HPRP.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROJECT BACKGROUND

For this thesis, the ultimate goal was to conduct a proof-of-concept test flight of the NPS SSAG HPRP launch vehicle at the FAR Rocket Range. To accomplish this goal, all three civilian amateur certification levels were first obtained by the author to establish an educational foundation in high-powered rocketry. To obtain certification, a total of three launches were conducted at the local TRA chapter, Tripoli Central California in Fresno, CA. Once certification was achieved, a test launch was conducted at the FAR Rocketry Range to familiarize the author and SSAG staff with the range's operational and logistical procedures for high-powered rocketry activities. The purpose of this chapter is to summarize each launch and the lessons learned throughout each certification level and the first NPS SSAG FAR launch. The lessons learned set the foundation for the NPS SSAG HPRP operational considerations and established the technical details of launch vehicle design.

A. LEVEL 1 CERTIFICATION (FLIGHT 1)

Level 1 certification was obtained at the Central California Chapter of the TRA in Fresno, CA on May 19, 2018. The airframe of the rocket used for certification was the commercially available HI-TECH^{TM} rocket kit made of phenolic tubing and produced by LOC Precision Rocketry. The rocket is a three-fin design, stands approximately four ft. (1.2m) tall, and has a dry weight of 1.3 lbs. (0.59 kg). The rocket has a 1.5 in. (38 mm) motor mount capable of accommodating a J-impulse class motor and a relatively low-performance ogive nose cone. The rocket carries no onboard electronics and the ejection system functions on a delay charge setup. The motor used for certification was the H128 reload kit produced by AeroTech Rocketry with an average thrust of 29 lbf (128 N), total impulse of 39 lbf-s (173 N-s,) and burn time of 1.3 s [45].

1. Certification Flight

As described in Table 6, the requirement for level 1 high-powered rocketry certification is the successful flight and recovery of a level 1 size rocket. The certification flight lasted approximately two minutes in total and the rocket achieved an approximate

altitude of 1,000 ft. (305 m) as indicated by visual reference and pre-flight simulation estimations. The rocket's one parachute functioned nominally near apogee and the rocket fell under parachute to the ground approximately 300 ft. (91 m) from the launch pad. Though there were minor burn marks on the parachute, the rocket sustained no structural damage from either launch or recovery and certification was achieved following the required post-flight inspection. Figure 6 depicts the rocket in its recovered configuration. At the forefront of the image is the main body of the rocket. The parachute and orange Nomex parachute protector can be seen in the upper right-hand portion of the image. The nose cone and attached payload section are at the end of the main airframe. All sections of the rocket are connected via the white, elastic parachute shock cord in the middle of the image. As evidenced by the image, the rocket sustained no damage during any portion of the flight and is ready for subsequent flight.

Figure 6. Level 1 Certification Rocket Landing and Recovery

2. Lessons Learned

Several lessons were learned from the successful flight and recovery of the level 1 certification rocket:

1. The “Z-fold” parachute packing technique can mitigate the possibility of riser entanglement upon parachute ejection and inflation. The method involves carefully preparing the parachute for packing and is described and pictured in the illustrated guide Modern High-Power Rocketry 2 [46].

2. Black powder parachute ejections are violent and can produce substantial thermal effects. Extra consideration for the thermal insulation in addition to a parachute protector must be considered to prevent burn holes in the canopy. In addition, Nomex shock cord protectors can be added for shock cord materials that require added heat resistance.

3. Construction of rocket motor reload systems is a detailed process. Any slight mistake during the process can result in catastrophic failure of the rocket motor and cause rapid disassembly of the rocket components. Therefore, the steps taken when putting the motor together should be executed in a rigorous, checklist-style fashion to ensure all procedures are done thoroughly and correctly.

4. Vent holes in the rocket airframe are necessary to prevent the pressure differential between the inside of the rocket and outside atmosphere from prematurely separating the nose cone during the rocket’s ascent. Vent hole size and placement is dependent of the specific airframe and rocket flight profile. Details on appropriate vent hole sizing and spacing can be found in [47].

5. Shear pins can be added to aid in sectional integrity of the rocket during flight. They are used to prevent the rocket from prematurely separating during ascent due to the pressure differential between the inside of the rocket and outside atmosphere. The ejection charge is then designed to be

significant enough to shear these pins and separate the sections of the rocket for parachute deployment. Ejection charges must be ground tested to ensure they generate enough force to shear each section's shear pins. Figure 7 is a computer aided design (CAD) rendering of an example of a shear pin's location and function in the rocket. In this example, the section coupler is epoxied to the main airframe of the rocket. The nose cone is then attached to the main airframe via the section coupler and secured with a shear pin.

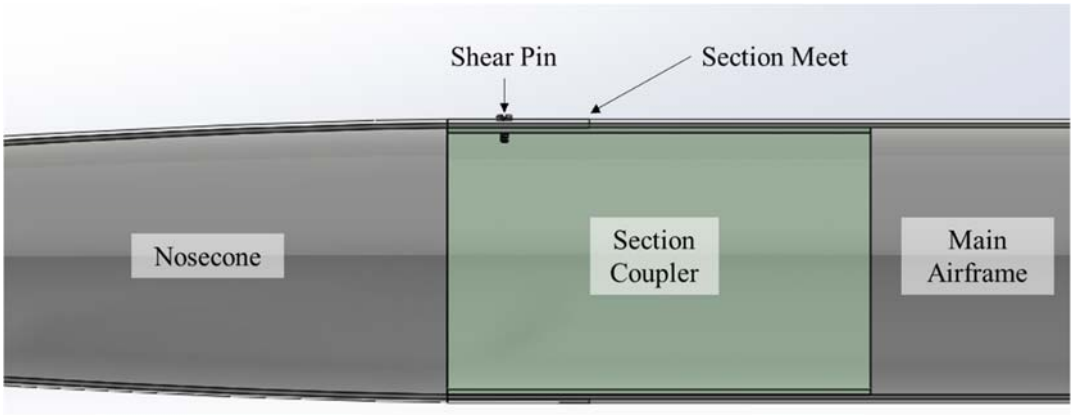


Figure 7. Shear Pin Example

B. LEVEL 2 CERTIFICATION (FLIGHT 2)

Level 2 certification was obtained at the Central California Chapter of the TRA on May 19, 2018. The airframe of the rocket used for certification was the commercially available EZI-65 rocket kit made of phenolic tubing and produced by LOC Precision Rocketry. The rocket is a three-fin design, stands approximately five ft. (1.5 m) tall, and has a dry weight of 2.5 lbs. (1.1 kg). The rocket has a 2.1 in. (54 mm) motor mount capable of accommodating a K-impulse class motor and has a relatively low-performance ogive nose cone. As with the level 1 certification rocket, the rocket parachute ejection system functions on a delay charge setup. The motor used for certification was the J90 reload kit produced by AeroTech Rocketry with an average thrust of 20 lbf (90 N), total impulse of 159 lbf-s (707 N-s), and burn time of 7 s [48]. In addition, the rocket carried

the initial prototype of a custom flight sensor package that recorded accelerometer and gyroscopic data. Though there are commercially available sensor packages that can do many of the desired functions of the custom sensor, building a sensor package allows for complete control of the rocket and payload while minimizing equipment cost. Such features would be important for enabling future capabilities of the NPS SSAG high-power rocket to include thrust vector control and high-fidelity telemetry and for expanding the educational concepts of the program.

The following is a list of the components used for the custom flight sensor package:

- Raspberry Pi (RPi) Zero (W) Microcomputer
- Analog Devices ADXL 377 ±200 g Accelerometer
- STMicroelectronics L3GD20H Triple-Axis Gyro
- Texas Instruments ADS7828 12-Bit, 8-Channel Sampling Analog-to-Digital Converter with I^2C Interface
- Texas Instruments UCC383-ADJ Low-Dropout 3A Linear Regulator
- STMicroelectronics LE33 3.3V Low-Dropout 100mA Linear Regulator

The RPi Zero microcomputer was programmed in Python and designed to continuously sample inputs from the gyroscope and accelerometer via a continuously running while loop. Data collected during the flight was stored on the RPi's secure digital (SD) card for post-flight processing. Figure 8 shows the lower portion initial flight sensor prototype design that flew on the level 2 certification rocket. In the image, the RPi microcomputer sits above the linear regulator on the pictured standoffs. The RPi is connected to the various sensors and status light emitting diodes (LED) via the header in the left-hand portion of the image.

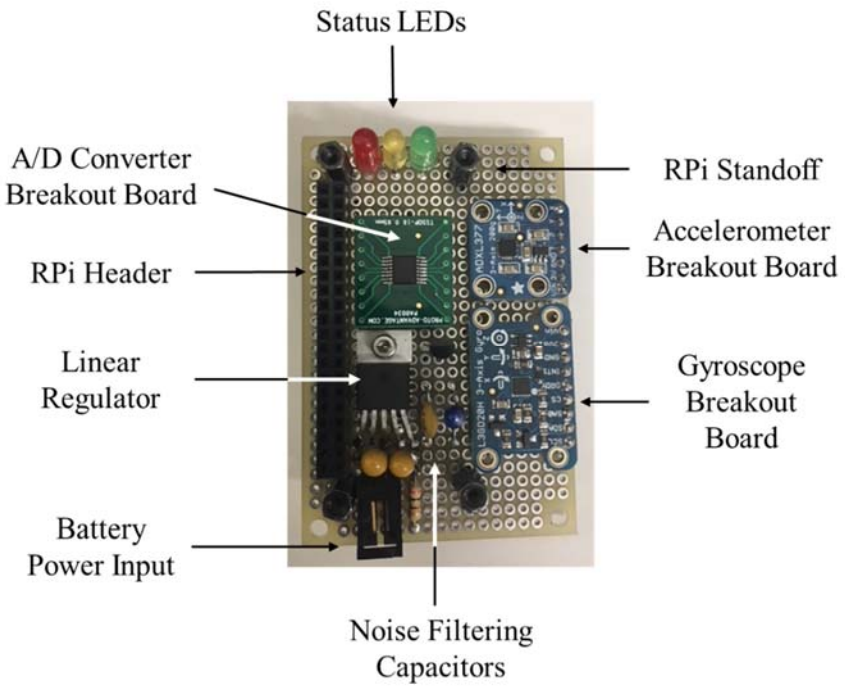


Figure 8. Initial Flight Sensor Package Prototype Design

During development of the prototype design, current draw for the sensors and RPi microcomputer was determined experimentally to be 335 mA. At an estimated flight operational time of 45 minutes for launch and recovery, the total current capacity of the battery pack powering the package needed to be 251 mA-h. Examination of Duracell's online technical library indicated that a battery pack consisting of four Duracell AA Coppertop batteries with an approximate capacity of 2,500 mA-h would meet the current and power requirements of the package for the duration of the flight [49]. According to [49], such a capacity would allow for sensor operation for the estimated 45 minutes for launch and recovery with an approximate hour and a half margin for any delays. The four Duracell AA battery pack was constructed using metal leads attached with a parallel gap resistance welder and reinforced with Kapton tape. Figure 9 displays the battery pack design and its attachment to the prototype sensor board. The metal tabs welded on the battery pack that can be seen in the image connect the batteries in series and to the sensor board.

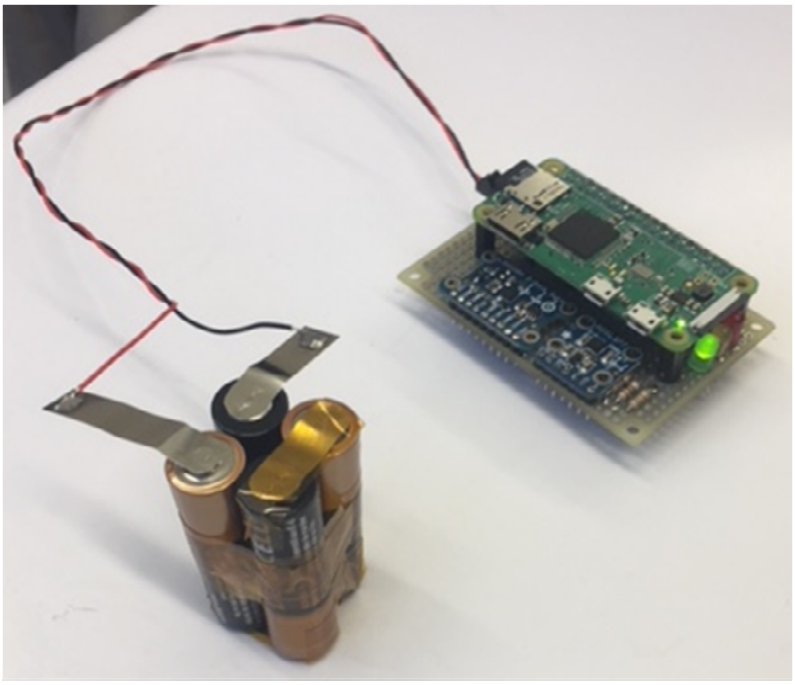


Figure 9. Sensor Prototype and Battery Pack Integration

1. Flight

The flight of the level 2 certification rocket lasted approximately four and a half minutes and the rocket achieved an approximate altitude of 5,200 ft. (1.6 km) as indicated by integration of the onboard accelerometer data. The rocket's one parachute functioned nominally near apogee and the rocket fell under parachute to the ground. Recovery of the rocket took approximately one and half hours as the rocket drifted under canopy into a large cornfield, making visual contact difficult. The rocket sustained no damage from either launch or recovery and certification was achieved following the required post-flight inspection.

2. Lessons Learned

Several lessons were learned from the successful flight and recovery of the level 2 certification rocket:

1. Onboard tracking is critical for future successful rocket recoveries. It was a matter of luck and determination that the level 2 rocket was recovered in the large corn field in which it drifted. Higher altitude flights will only drift farther from the launch site and require some augmented tracking capability for successfully recovering the rocket.

2. Operational integration of onboard sensors is a difficult problem that must be addressed prior to any flight. Among other things, careful consideration must be given to when the sensor will be placed into the rocket, how the sensor will be turned on, how the functionality of the sensor will be verified once it is inside the rocket, how the data will be recovered, and how the sensor will behave during non-nominal flight operations.

3. Flight sensor scales need to be paired with the expected launch condition environment. Using a 200g accelerometer to characterize an approximate maximum acceleration of 10g's induced significant noise into the acceleration data. Using a lower threshold accelerometer in conjunction with the 200 high-g accelerometer will increase accelerometer data fidelity.

4. Accelerometer and gyroscope data were not sufficient for fully characterizing the rocket's flight. More sensors such as a barometer, GPS, and camera are needed to correlate data during key flight events.

C. LEVEL 3 CERTIFICATION (FLIGHT 3)

Level 3 certification was obtained at the Central California Chapter of the TRA on August 18, 2018. The airframe of the rocket used for certification was the commercially available 4 in. (10 cm) fiberglass DX3 XL rocket kit produced by Madcow Rocketry. Shown in Figure 11, the rocket is a three-fin design, stands approximately seven and a half ft. (2.3 m) tall, and has a dry weight of 14 lbs (6.4 kg). The rocket has a 3.0 in. (75 mm) motor mount capable of accommodating an M-impulse class motor and a high-performance metal-tipped Von Kármán style nose cone. To prevent excessive drift

under canopy, the rocket employs a “dual-deployment” recovery technique in which a small drogue parachute deploys at apogee while the main parachute deploys at 1,500 ft. (460 m). Such a technique allows the rocket to fall faster from apogee while maintaining a safe velocity for main parachute deployment. This prevents excessive drift in the presence of wind while slowing the rocket sufficiently for a safe landing. Figure 10 shows the planned flight profile of the rocket. In the image the dual-deployment technique can be visualized as the rocket’s drogue parachute deploys at apogee and the main parachute deploys at 1,500 ft. (460 m). Ejection charge timing is controlled by onboard flight electronics and flight data is collected via the custom sensor package. The motor used for certification was the M650 reload kit produced by AeroTech Rocketry with an average thrust of 146 lbf (650 N), total impulse of 1,350 lbf-s (6,000 N-s), and burn time of 9 s [50]. Figure 11 shows the fully assembled level 3 certification rocket. Sections of the rocket are annotated to show how the rocket was configured during the certification launch attempt. The rocket’s drogue parachute was placed in the lower airframe section of the rocket and the main parachute was placed in the payload section. During nominal operations, the rocket is separated into three different sections once all the parachutes are deployed. Detailed documentation of the rocket, its mission profile, and the construction process are available in [51].

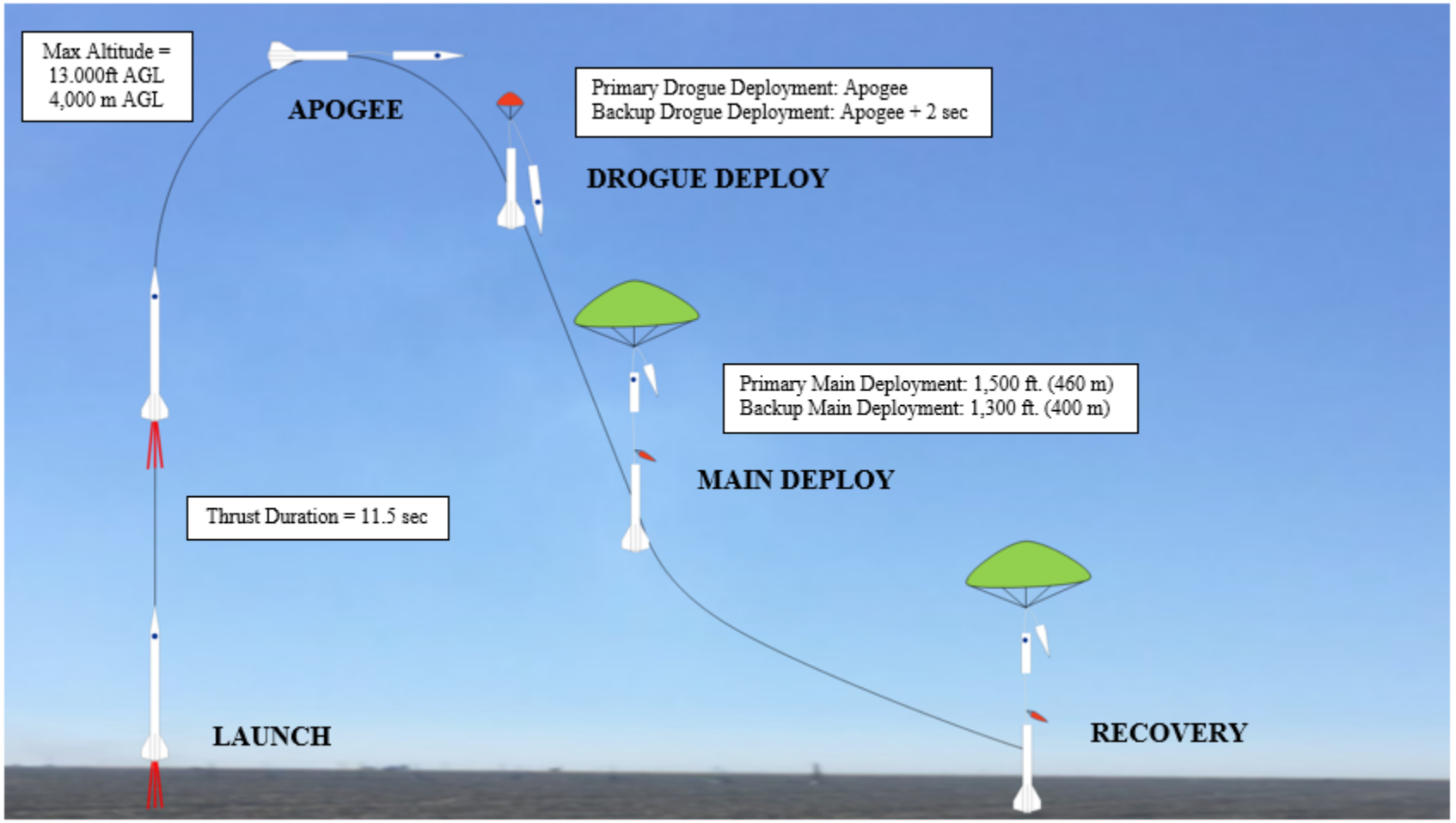


Figure 10. Planned Flight Profile of Level 3 Certification Rocket

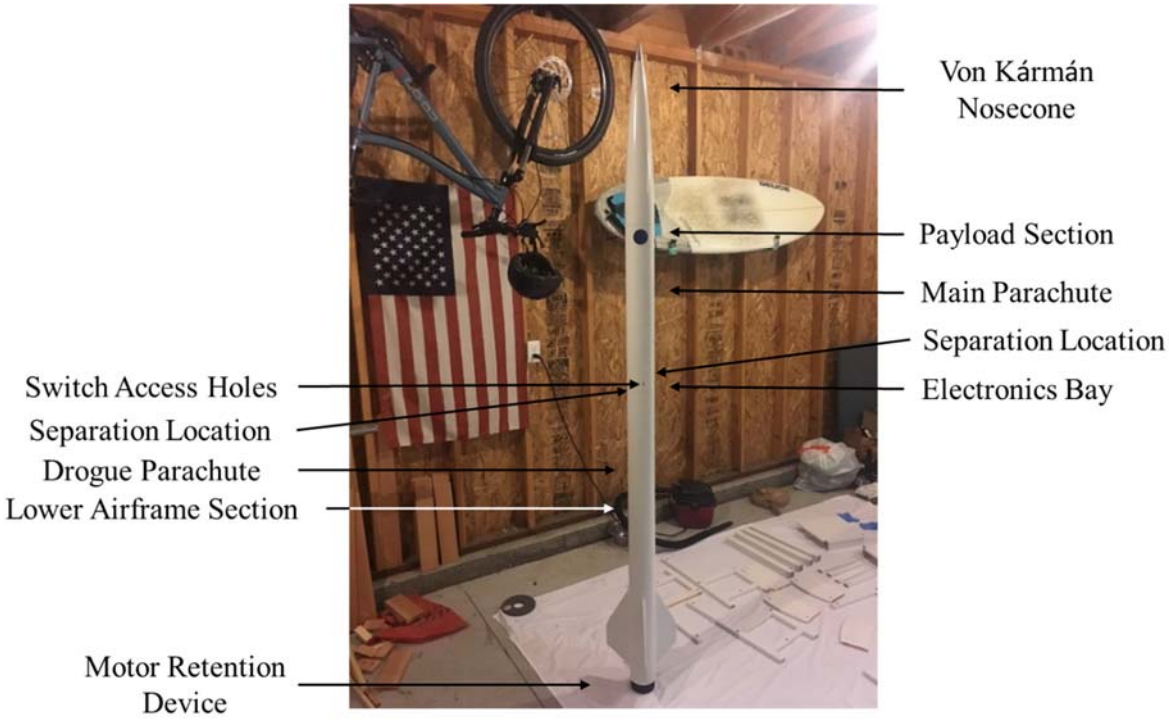


Figure 11. Fully Assembled Level 3 Certification Rocket

Post-flight data analysis of the level 2 rocket indicated that the sensors on the custom flight sensor package were insufficient for fully characterizing the rocket's flight. The following additional sensors were added in attempts to better understand the level 3 rocket's launch environment and characterize its flight:

- Xtrinsic MPL3115A2 I^2C Precision Altimeter
- Raspberry Pi Camera v.2
- GlobalTop Technology Inc. FGPMMOPA6H Global Positioning System (GPS) Standalone Module

The two flight computers that were integrated to control ejection system functions for the rocket were the commercially available EasyMini and StratoLogger (compact footprint version) produced respectively by Altus Metrum and PerfectFlite. Additionally, a 70cm 100mW GPS/Automatic Packet Reporting System (APRS) transmitter produced

by BigRedBee was added to the rocket for tracking due to lessons learned during the level 2 certification launch (B 2.1). Figure 12 shows the flight electronics that were mounted inside the rocket's electronics bay. The white mounting sled is a custom designed 3D-printed polycarbonate piece that secures all of the components during launch. Mechanical switches produced by FingerTech were added to turn on the flight sensors once they are mounted inside the rocket. The switches can be accessed via holes in the rocket airframe with a hex wrench prior to launch.

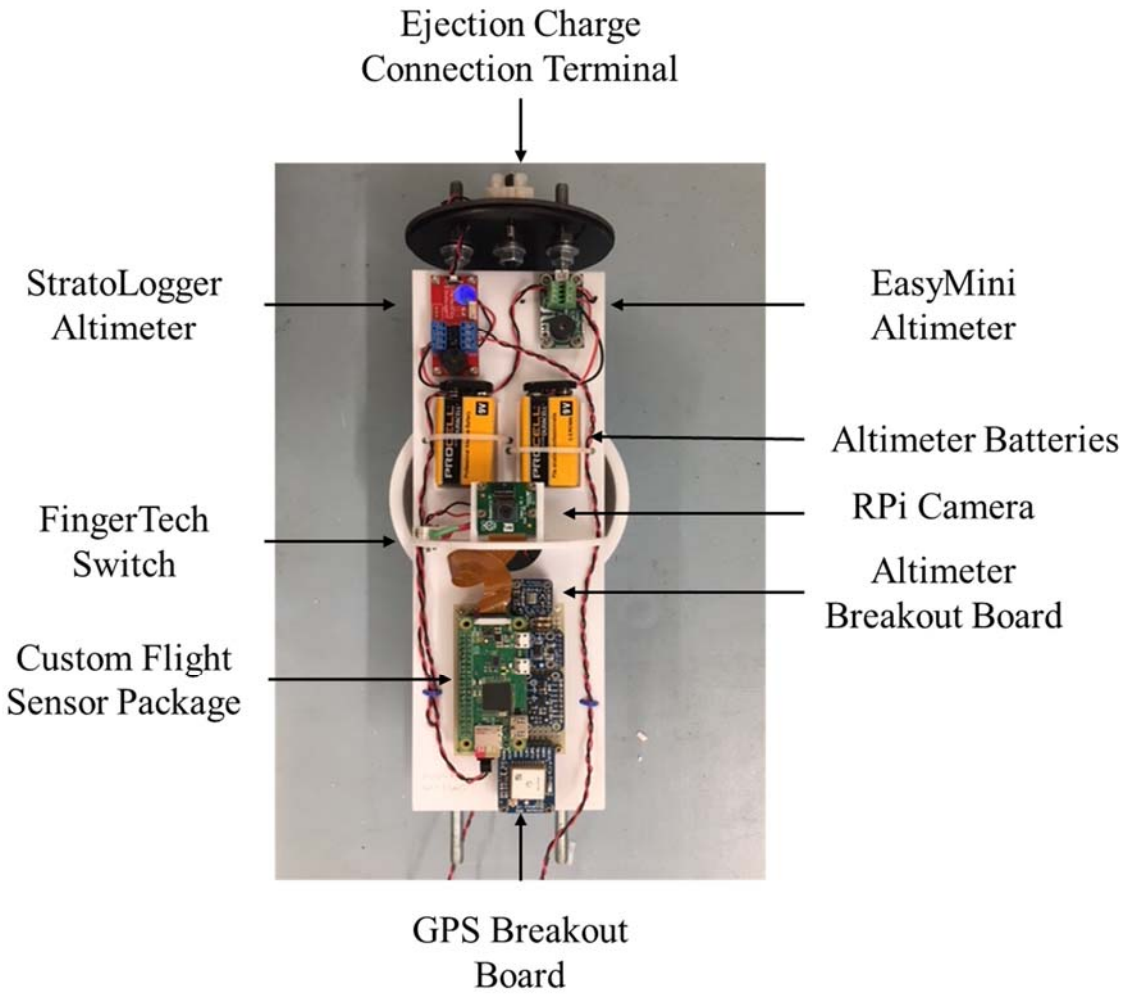


Figure 12. Level 3 Certification Rocket Flight Electronics

1. Pre-flight Analysis

Key flight performance parameters for the level 3 certification attempt were calculated using both MATLAB and RockSim. Table 8 summarizes the results of the analysis. Examination of the calculated values indicated that the maximum projected height of the rocket, as calculated by both Rocksim and the author, varied by 3%. Furthermore, both values indicated that the rocket would remain within the FAA waiver altitude of 16,800 ft. (5,120 m) AGL for the Tripoli Central California launch site at its altitude of 150 ft. (55 m) mean sea level (MSL).

Table 8. Flight 3 Pre-flight Analysis Summary2

Prediction Method	Maximum Acceleration	Maximum Velocity	Maximum Height - AGL	Time to Apogee
RockSim Calculated	351 ft./s^2 107 m/s^2	1,060 fps 324 m/s	13,400 ft. 4,080 m	27 s
Author Calculated [51]	345 ft./s^2 105 m/s^2	1,030 fps 313 m/s	13,800 ft. 4,200 m	28 s

2. Flight

The flight of the level 3 certification rocket lasted approximately 13 minutes and the rocket achieved an altitude of about 14,600 ft. (4.5 km) AGL as indicated by the BigRedBee GPS data. At launch, the launch guide rail was of insufficient length for the rocket to achieve stable flight before departing the guide rail, inducing a deviation from a vertical flight path. Chapter IV Section C discusses passive stability of rockets and explains why the rocket must have sufficient velocity before departing the guide rail to achieve stable, vertical flight. Figure 13 shows the rocket during the initial stages of its

[2] RockSim key flight performance parameter calculations include a variable coefficient of drag parameter based on velocity from a proprietary database of experimental measurements. In contrast, the author's MATLAB script uses an average number for the coefficient of drag. Such a difference can produce the results shown in Table 8 where even though the author's calculated maximum velocity is lower than RockSim's estimate, the author's estimated maximum height is higher.

boost phase. The smoke trail from the rocket's exhaust in the image highlights the approximately 20° deviation from vertical.

Despite the approximate 70° launch elevation angle, the rocket's drogue parachute functioned nominally near apogee and the rocket fell to an altitude of 1,500 ft. (460 m) before the main parachute deployed. Recovery of the rocket took approximately one hour as the rocket was roughly three miles (4.9 km) from the launch site and out of visual contact. Using GPS packets from the BigRedBee transmitter, the rocket was found in a large vineyard east of the launch site. The rocket sustained no damage from either launch or recovery and certification was achieved following the required post-flight inspection.

Figure 13. Level 3 Certification Launch

Due to a lack of understanding and testing of the custom sensor package's programming, the custom sensor package experienced an error that caused the loss of all data captured during the flight. Following post-flight analysis, it was determined that the

error was caused by a failure of the RPi camera which terminated the flight software script sometime after it was initiated and prior to the rocket's recovery. Through post-flight testing, it was determined that there were three likely causes of the camera error:

1. The ribbon cable connecting the RPi to the camera came loose.
2. The memory on the RPi reached capacity.
3. During the final system test, the flight software was not properly terminated and the RPi registered the camera as being allocated to another resource.

As there was no error handling for a camera failure, the flight software script module crashed. Termination of the script caused a complete data loss as the file containing sensor data was not being periodically saved. This was an oversight in software design and testing and was fixed for future rocket flights. Though the custom sensor data was a complete loss, flight GPS data was captured by the BigRedBee GPS/APRS transmitter. Figure 14 visually displays the data captured from the device as a keyhole markup language (KML) track on Google Earth. The track shows the non-vertical launch angle as well as the successful drogue and main parachute deployments by the dual-deployment ejection system.

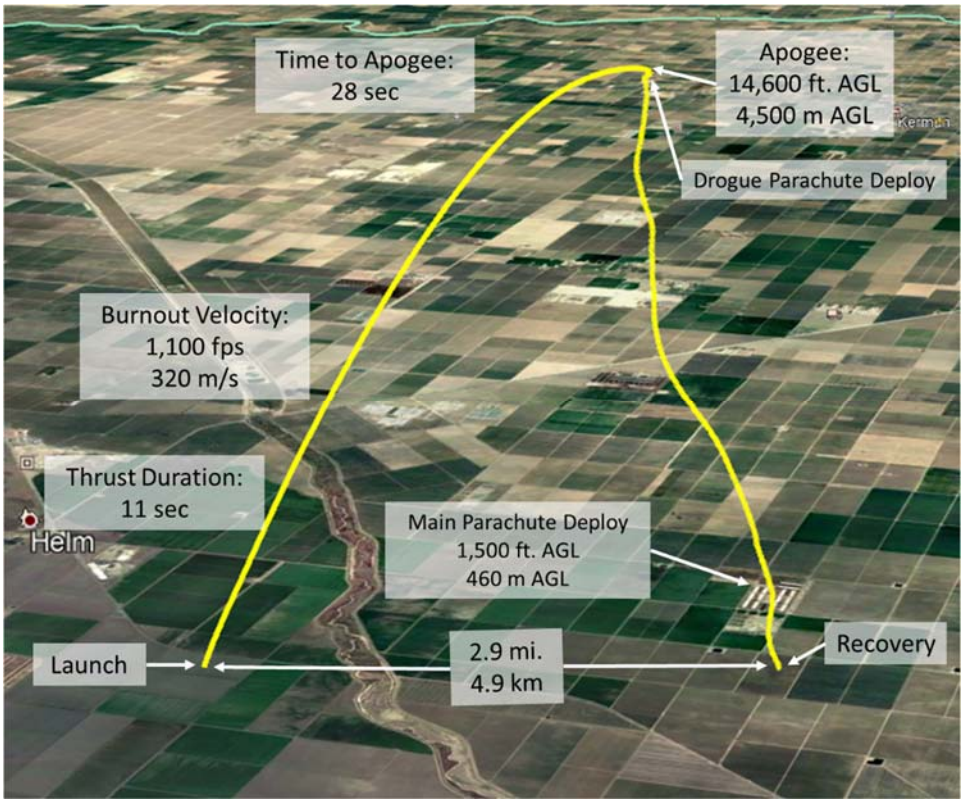


Figure 14. Level 3 Certification Rocket Flight KML File Rendering

3. Lessons Learned

Several lessons were learned from the successful flight and recovery of the level 3 certification rocket:

1. Launch rails must be long enough for the rocket to achieve stable flight prior to departure. Though the launch rail was the same standard length as used in several launches earlier in the day, it was not suitable for this particular combination of rocket motor thrust and rocket size. Minimum launch rail length for the rocket to achieve stable flight must be calculated and known for each individual launch. A typical launch rail exit velocity for stable flight is 45 fps (14 m/s) or approximately 30 mph. Using RockSim or the MATLAB script in Appendix A, the minimum launch rail size can be calculated for each individual launch based on this velocity.

Figure 15 is a plot of the flight 3 rocket's velocity as a function of guide rail length generated during post-flight analysis. It can be seen from the graph that the rocket achieved the estimated point of stable velocity after departing the launch guide rail (annotated by the vertical red line).

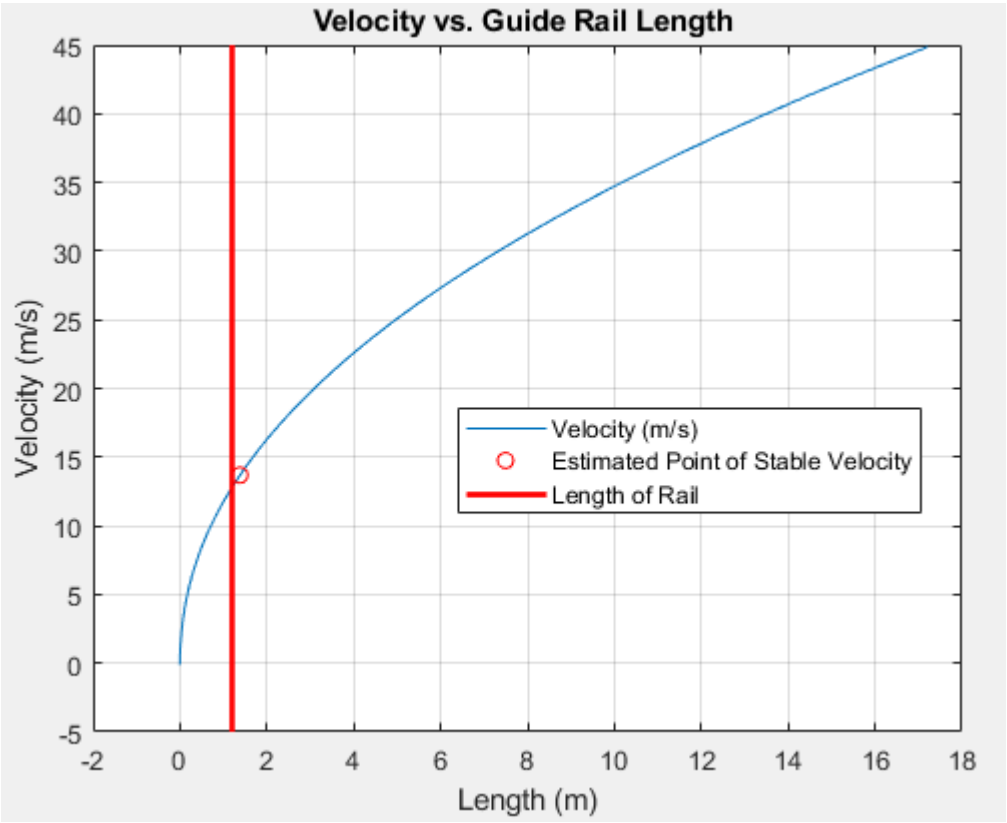


Figure 15. Guide Rail Length vs. Velocity for Level 3 Certification Launch

2. Error handling is one of the most important features of any flight software. Failure to properly handle errors caused by the RPi camera terminated the data collecting script and caused a complete loss of sensor data. Ensuring data is saved in small chunks throughout the duration of the flight, and errors are properly handled by the flight software, will provide the best chances of collecting data from each flight.

3. Telemetry is an important aspect of rocketry operations. Without the GPS telemetry packets from the BigRedBee sensor it is likely that the rocket would not have been found and no data from the flight would have been recovered. Therefore, future iterations of the custom sensor package must have a method of sending telemetry during the rocket's flight.

4. Independent cameras may be a better solution than using RPi cameras. While compact in form factor and relatively cheap, the RPi camera poses several integration issues such as a limited ribbon cable length and software problems that complicate their incorporation into rockets. To mitigate the possibility of the camera module crashing the data capturing software, more robust error handling should be incorporated into the flight software or independent cameras such as a GoPro could be used.

D. FAR OPERATIONAL CHECKOUT LAUNCH (FLIGHT 4)

Having obtained the necessary civilian certifications and established the requisite rocketry knowledge, a FAR Rocket Range operational checkout launch was conducted on December 18, 2018. The purpose of the launch was to familiarize SSAG staff and students with the operations and logistics associated with a rocket launch at the FAR Rocket Range for future flights conducted as part of the NPS SSAG HPRP. The rocket used for the flight was the same level 3 rocket used for certification. The rocket had no major modifications in the airframe or operational design. The custom sensor package included a more robust software design that incorporated error handling and the motor used for launch was changed to the more powerful M1315. The M1315 reload kit produced by AeroTech Rocketry has an average thrust of 296 lbf (1,315 N), total impulse of 1,506 lbf-s (6,700 N-s), and burn time of 5.4 s [52]. The motor has a higher initial thrust than the M650 certification motor which mitigated the possibility of sub-optimal launch angle as seen with the level 3 certification attempt.

The launch rail used for this launch was the "Newman 10-Foot" created by John Newman of the FAR Rocket Range. The 1/4" (0.654 cm) t-slot aluminum extrusion rail is 10 ft. (3.1 m) in length and reinforced with steel support. The rail was chosen for its

adequate length (Chapter III Section C 2.1 lesson learned) and slot size, which was sufficient for accommodating the rocket's 1010 size rail buttons. Figure 16 depicts the Newman 10-Foot launch rail in one of its angled-launch configurations.

Figure 16. Newman 10-Foot Launch Rail. Source: [53].

Figure 17 shows a graph of the rocket's velocity as a function of the guide rail length. It can be seen from the plot that the 35 lb. (16 kg) rocket achieves the estimated stable velocity of 45 fps (14 m/s) at a location of approximately 2.8 ft. (0.85 m) on the guide rail after launch. Given that the rail used for this particular launch was 10 ft. (3.1 m), the rocket would have an approximate 79% margin for achieving a stable flight velocity before departing the rail.

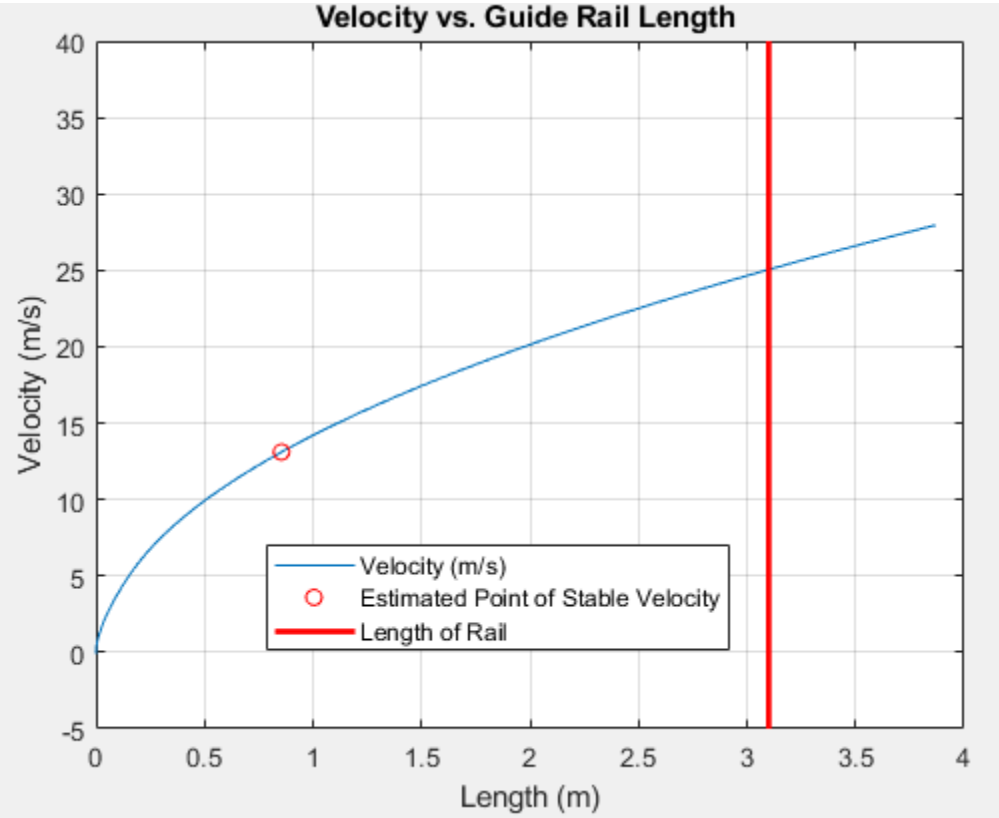


Figure 17. Guide Rail Length vs. Velocity for FAR Operational Checkout Launch

1. Pre-flight Analysis

Key flight performance parameters for the FAR operational checkout launch were calculated using both MATLAB and RockSim. Table 9 summarizes the results of the analysis. Examination of the calculated values indicated that the maximum projected height of the rocket, as calculated by both Rocksim and author, varied by 4%.

Table 9. Flight 4 Pre-flight Analysis Summary

Prediction Method	Maximum Acceleration	Maximum Velocity	Maximum Height - AGL	Time to Apogee
RockSim Calculated	453 ft./s^2 138 m/s^2	1,450 fps 442 m/s	17,200 ft. 5,242 m	30 s
Author Calculated	332 ft./s^2 101 m/s^2	1,180 fps 358 m/s	17,900 ft. 5,480 m	31 s

2. Flight

The flight of the FAR operational checkout rocket lasted approximately 1 minute, 15 seconds and the rocket achieved an approximate altitude of 17,700 ft. (5.39 km) AGL as indicated by the BigRedBee GPS data as shown in Figure 18. At apogee, the drogue parachute failed to deploy, and the rocket followed a ballistic trajectory. During the search for the rocket, the nose cone was found approximately 1,000 feet (300 m) from the main airframe of the rocket with a sheared parachute shock cord line. This indicates the main parachute and nose cone ejection functioned nominally at 1,500 feet (460 m) and the forces associated with parachute deployment separated the nose cone, but the high speed of separation resulted in forces that sheared the cord connecting the nose cone to the upper airframe section. Through post-flight analysis, it was determined that there were four likely causes of the drogue parachute ejection failure acting either singularly or in concert:

1. The black powder ejection charges in the drogue parachute compartment were not sufficient to shear the retention pins and separate the sections.
2. The rocket's forward velocity of approximately 164 fps (50 m/s)[3] at apogee produced too much pressure on the nose cone and prevented the lower airframe of the rocket, which contained the drogue parachute, from separating from the electronics bay at apogee.
3. The nose cone was not properly aligned during assembly causing binding between the two separating sections.
4. Shear pin indexing was not accurate enough to ensure symmetrical loading of the pins during ejection. This could have caused one or more shear pins to remain intact during the separation phase.

[3] Estimated using interpolation of BigRedBee GPS data points on either side of the rocket's apogee..

Figure 18 visually displays the data captured from the BigRedBee tracking device as a KML track on Google Earth. The track confirms the ballistic profile of the rocket's flight and the failure of the lower airframe separation and drogue parachute deployment at apogee. Figure 19 displays a truncated portion of the KML track on Google Earth of the final seconds before the rocket's impact. The deviation in the ballistic track annotated in the image confirms the separation of the section's nose cone and subsequent deployment of the main parachute. The BigRedBee device survived the fall in the nose cone after separation and shock cord failure and continued to transmit data, which was detected during recovery operations when the recovery team approached within a few hundred feet of the device.

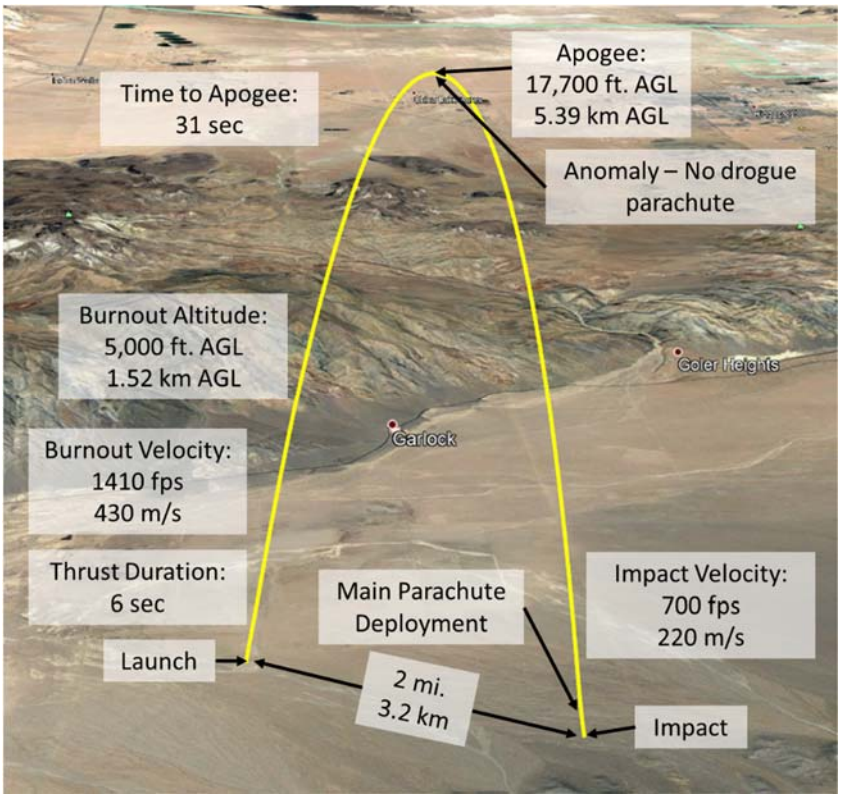


Figure 18. FAR Operational Checkout Rocket Flight KML File Rendering

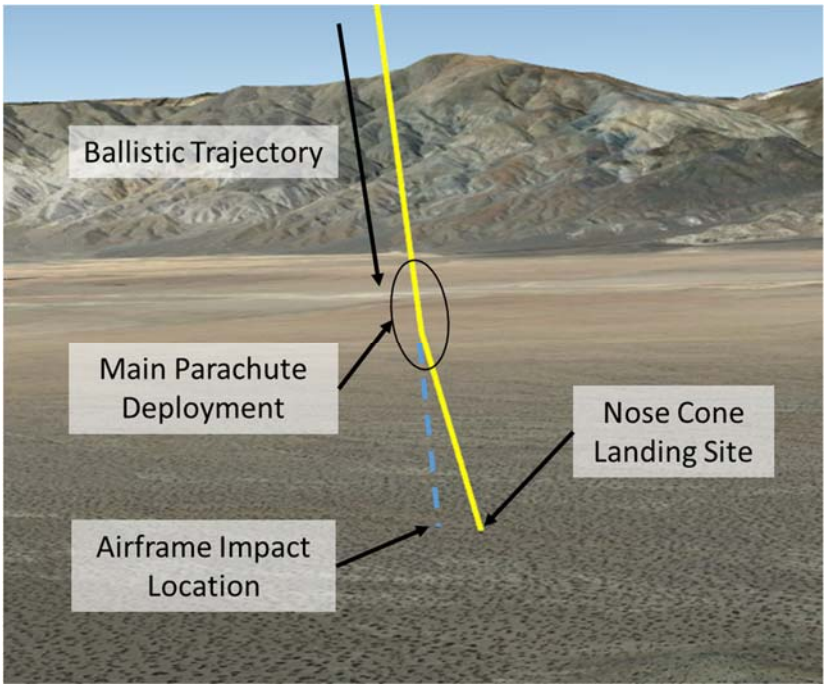


Figure 19. FAR Operational Checkout Rocket Flight KML File Rendering - Impact of Rocket

Figure 20 displays the rocket's nose cone as it was recovered by the recovery team. Of note is the sheared main parachute shock cord in the upper middle portion of the image. The image further confirms the main parachute deployment event that generated forces great enough to shear the 1,500 lbf (6,700 N) test strength braided Kevlar shock cord. Figure 21 displays the remainder of the rocket's airframe that continued on the ballistic trajectory and impacted the ground. The image confirms the high velocity of impact as only about a foot of the rocket is visible in the image while the remaining approximate five feet of the airframe is buried in the sand. The depth of the crater created by the impact is shown in Figure 22 which depicts the author standing in the hole that was dug to recover all of the components of the buried airframe.

Figure 20. FAR Operational Checkout Nose Cone Recovery

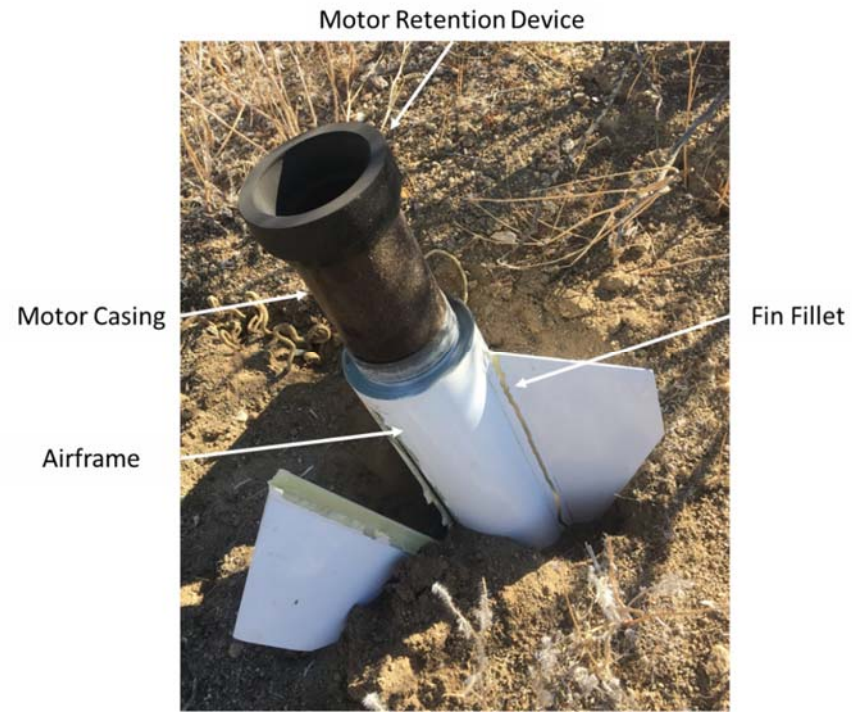


Figure 21. FAR Operational Checkout Airframe Recovery

Figure 22. FAR Operational Checkout Airframe Crater

3. Lessons Learned

Several lessons were learned from the flight and post-anomaly analysis of the FAR operational checkout rocket:

1. Friction between separating portions of the rocket can increase the chances on binding during parachute and payload ejections. All separating sections should be smoothed and lubricated to aid in separation and prevent accidental binding.

2. According to [54], incomplete combustion of unconstrained black powder may occur in high-altitude rocket flights. Consequently, the effectiveness of black powder ejection charges at altitude may not match that of the ground test. Therefore, all black powder ejection charges should be padded with margin and tightly packed and sealed to facilitate the completeness of the black powder combustion. Preferably, high-altitude

rockets should forgo black powder as an ejection charge and use another method such as CO_2 ejection as a cleaner, more reliable ejection method.

3. The layout of shear pins and the proper index of their placement is critical to ensuring proper rocket segment separation. Uneven, or misaligned shear pins may create inconsistencies in the separation force and prevent all pins from shearing.

4. Backup ejection charges should be significantly greater than the primary ejection charge. While the primary ejection charge should be sized to separate each section of the rocket with the minimal energy necessary to prevent damage, backup charges should be significantly greater to ensure separation. The backup charge in this rocket was only 10% greater than the primary and did not meet the intended purpose of providing a backup safety for the rocket's ejection system. Based on ground testing, it is recommended that a minimum 50% margin be added to the backup charge to ensure separation for future flights.

5. Shovels should be included in the rocket recovery kit. Even if the rocket appears to have had a nominal parachute deployment and descent, taking a shovel to the landing or impact site will save time and aid in the recovery of all the rocket's components.

E. POST-CHECKOUT LAUNCH CONCLUSIONS

The lessons learned during the certification and FAR operational checkout launches set the educational foundation and baseline for the design of the standard launch vehicle platform and operational procedures for the SSAG HPRP. Each rocket build and subsequent launch produced valuable insight into amateur rocketry and its potential to serve as an education tool and payload delivery platform for the NPS SSAG HPRP. From the successful airframe recovery in three of the four launches, it was evident that it was possible to reduce program recurring cost through the standardization of a single, robustly designed launch vehicle platform.

To achieve altitudes that would complement the HAB project (i.e., altitudes exceeding 120,000 ft. (37 km)), with amateur high-power rockets that utilized commercial-off-the-shelf motors, staging would be necessary. Though staging undoubtedly increases launch vehicle design and operational complexity, it greatly increases the mission altitude profile for a single launch vehicle. Much like the NSRP, a single launch vehicle with multiple variants will enable large mission variability at a relatively low cost for the NPS SSAG HPRP.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. NPS SSAG HIGH-POWER ROCKET

The next step in the program was the creation of a launch vehicle that would serve as the foundation of the NPS SSAG HPRP. Based on the four flights described in Chapter III, the conceptual design of the NPS SSAG high-power rocket was driven by the desire for a launch platform that enables high mission variability and quick mission turnaround while maintaining a low operational cost. The initial manifestation of this concept was a large, two-stage, unguided rocket designed similarly to the U.S. meteorological sounding rocket, the Super Loki Dart [55].

Shown in Figure 23, the Super Loki Dart is a two-stage meteorological sounding rocket developed for use by the United States Air Force. The rocket is capable of conducting meteorological measurements between the altitudes of 66,000 – 300,000 ft. (20 – 90 km) [55]. Since its inception in 1969, the Super Loki Dart rocket design has proved both robust and stable. Hundreds of launches conducted with multiple variants of the Loki design have enabled meteorological research for various government and contracted agencies [56]. The design was chosen for the basis of the NPS SSAG high-power rocket because of its relative low cost and proven flight history.

The purpose of this chapter is to describe the technical details associated with the NPS SSAG high-power rocket. The chapter will start with a summary of the rocket's characteristics and the operational design of the rocket's flight profile. Following an analysis of the rocket's stability, each subcomponent of the rocket is discussed in detail.

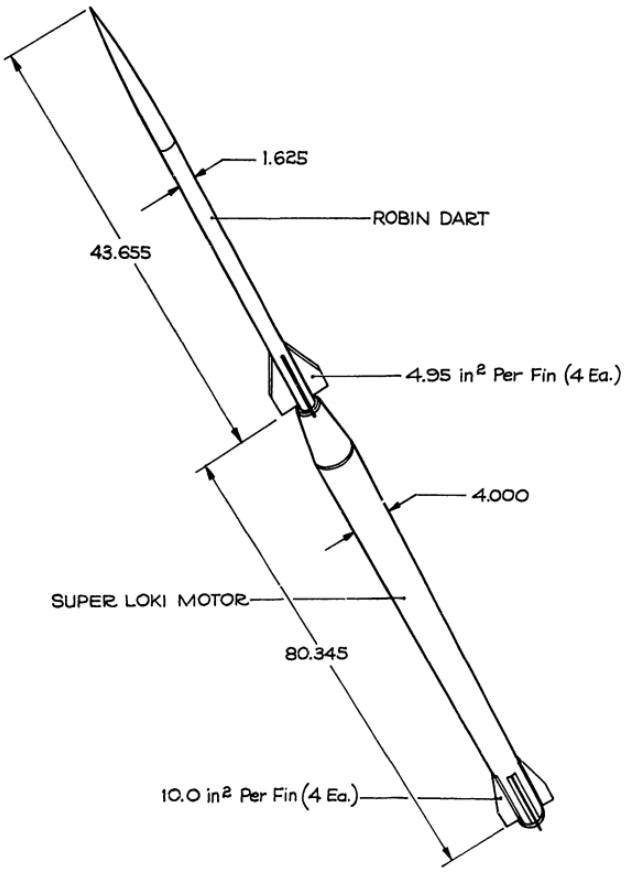


Figure 23. Super Loki Dart Launch Vehicle. Source: [55].

A. ROCKET CHARACTERISTICS

The overall two-stage configuration stands 21.5 ft. (6.55 m) tall, weighs approximately 175 lbs. (79.4 kg) at launch, and has a total installed impulse of 25,830 lbf-s (114,900 N-s). The rocket is designed to achieve altitudes between 20,000 and 250,000 ft. (6 – 76 km) and deploy a four-lb. (1.8 kg) CubeSat payload at apogee. Figure 24 is a scale drawing of the rocket in its two-stage configuration. Of note in the figure is the larger-diameter nose cone, which carries the CubeSat payload. While the rest of the rocket is a performance-optimized minimum-diameter design, the transition and larger nose cone were necessary to accommodate the dimensions of a CubeSat. The feature is further highlighted in Figure 25 which depicts a CAD rendering of the two-stage rocket configuration.

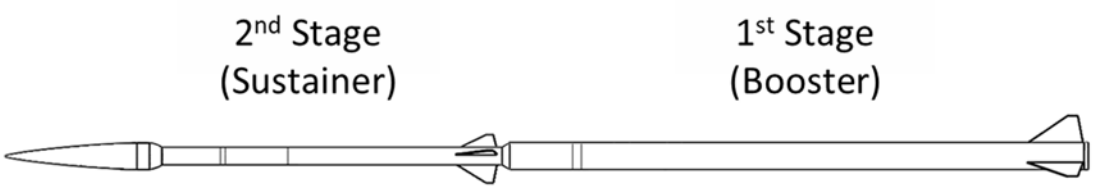


Figure 24. Scale Drawing of the NPS Class-3 High-Power Rocket

Figure 25. CAD Rendering of Rocket Two-Stage Configuration

To optimize the rocket's altitude, the 1st stage booster is a 6 in. (152 mm) diameter three-fin design capable of accommodating a 6 in. (152 mm) diameter motor. In addition, the stage consists of an aluminum stage transition piece, upper airframe section, electronics bay, and main body tube. The booster-stage motor is the commercially available Q15781, produced by Animal Motor Works (AMW) in conjunction with

Cesaroni Technologies Incorporated (CTI). The Q15781 motor has a total impulse of 21,100 lbf-s (93,900 N-s) and a burn time of 5.95 seconds and is capable of propelling the combined two-stage rocket to a velocity of 3,700 fps (1,100 m/s or Mach 3.2) [20]. For recovery, the booster stage employs a dual-deployment technique and is equipped with two independent and redundant ejection systems. Figure 26 is a CAD rendering of the booster stage and aforementioned components.

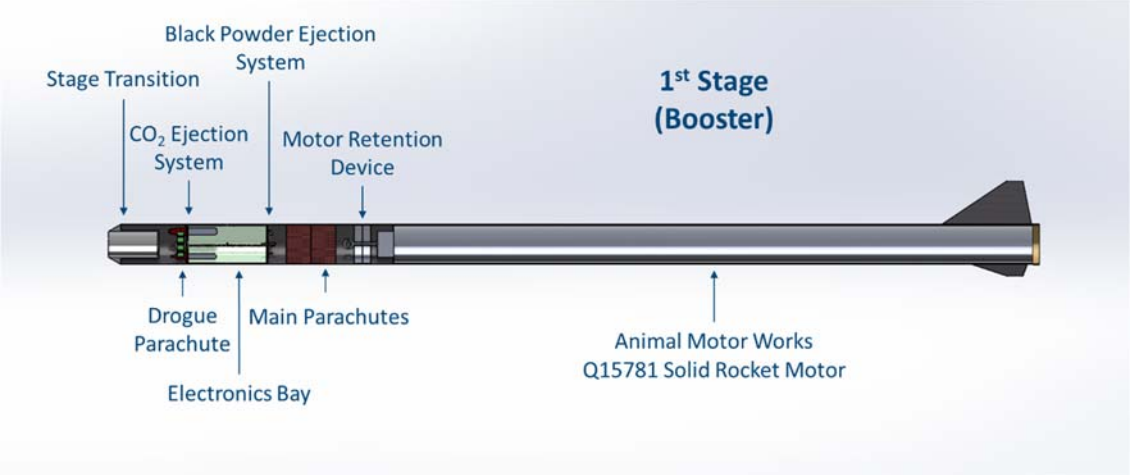


Figure 26. CAD Rendering of Rocket 1st Stage Booster

The sustainer stage of the rocket is a 4 in. (102 mm) diameter three-fin design capable of accommodating a 3.9 in. (98 mm) motor and comprises a 6 in. (152 mm) metal-tipped LD-Haack "Von Kármán"–style nose cone, 2U CubeSat payload, aluminum transition piece, payload section, electronics bay, and main body tube. The sustainer-stage motor is the commercially available O3400 motor produced by CTI. The O3400 motor has a total impulse of 4,700 lbf-s (21,000 N-s), a burn time of 6.3 seconds, and is capable of carrying the sustainer stage to an apogee altitude of approximately 250,000 ft. (76 km). For recovery, the sustainer stage employs a dual-deployment technique and is equipped with two independent and redundant ejection systems. Figure 27 depicts a CAD rendering of the sustainer stage and components.

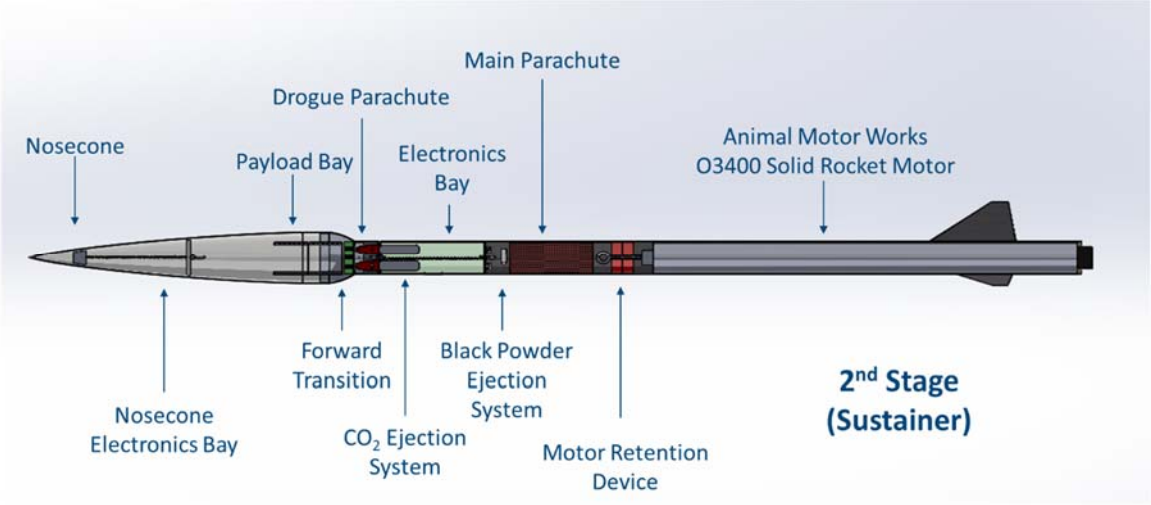


Figure 27. CAD Rendering of Rocket 2nd Stage Sustainer

B. PLANNED FLIGHT PROFILE

Being a two-stage rocket design, the NPS SSAG high-power rocket is capable of flying two variable flight profiles depending on the mission requirements. As seen with the NSRP launch vehicles, a two-stage design increases mission variability and permits a wider range of launch altitudes than a single stage launch vehicle. By changing the motor used in both the booster and sustainer, the rocket is capable of achieving altitudes between 20,000 and 250,000 ft. (6 – 76 km).

The first flight profile, depicted in Figure 28, is a single-stage launch of the sustainer section. In this configuration, the rocket is capable of carrying a four lb. (1.8 kg) CubeSat payload to a maximum altitude of approximately 32,000 ft. (9.8 km) AGL in about 37 seconds. The CubeSat payload is deployed at apogee of the flight along with the drogue parachute of the sustainer airframe. The CubeSat descends separately under its own parachute, as do the nose cone and sustainer airframe. Finally, at 1,500 ft. (460 m) AGL, the main parachute of the rocket is deployed, slowing the rocket to a lower, safer velocity for landing. Such a flight profile is useful for lower-altitude rocket and payload testing.

In the second configuration of the rocket, depicted in Figure 29, the booster stage of the rocket is added. In this configuration, the rocket is capable of carrying a four lb. (1.8 kg) CubeSat payload to a maximum altitude of approximately 250,000 ft. (76 km) in about two and a half minutes. The CubeSat payload is deployed at apogee of the flight along with the drogue parachute of the rocket. As with the sustainer-only configuration, the main parachute of the rocket is then deployed at 1,500 ft. (460 m), slowing the sections of the rocket for landing. Such a profile is useful for high-altitude payload and rocket testing and research.

Key flight performance parameters for the rocket were calculated using the commercially available program RockSim and the author's MATLAB script named rocket flight simulation estimate, available in Appendix A. Two methods were used to calculate the key flight performance parameters to increase the confidence of the analysis for the untested system. Table 10 shows the results of the analysis.

Table 10. Key Flight Performance Parameters for NPS SSAG High-Power Rocket

Single-Stage Sustainer Launch (Burn time 6.3 s)				
Prediction Method	**Max Acceleration**	**Max Velocity**	**Max Altitude - AGL**	**Time to Apogee**
RockSim Predicted	727 ft./s^2 222 m/s^2	2,640 ft./s 804 m/s	32,000 ft. 9,750 m	38 s
Author Predicted [App. I Sec. B]	562 ft./s^2 171 m/s^2	2,160 ft./s 658 m/s	31,000 ft. 9,500 m	37 s
Two-Stage Full Mission Launch (Burn time 12.3 s – Total)				
Prediction Method	**Max Acceleration**	**Max Velocity**	**Max Altitude**	**Time to Apogee**
RockSim Predicted	863 ft./s^2 263 m/s^2	4,240 ft./s 1,290 m/s	244,000 ft. 74,000 m	138 s
Author Predicted [App. I Sec. A]	787 ft./s^2 240 m/s^2	4,620 ft./s 1,300 m/s	246,000 ft. 75,000 m	131 s

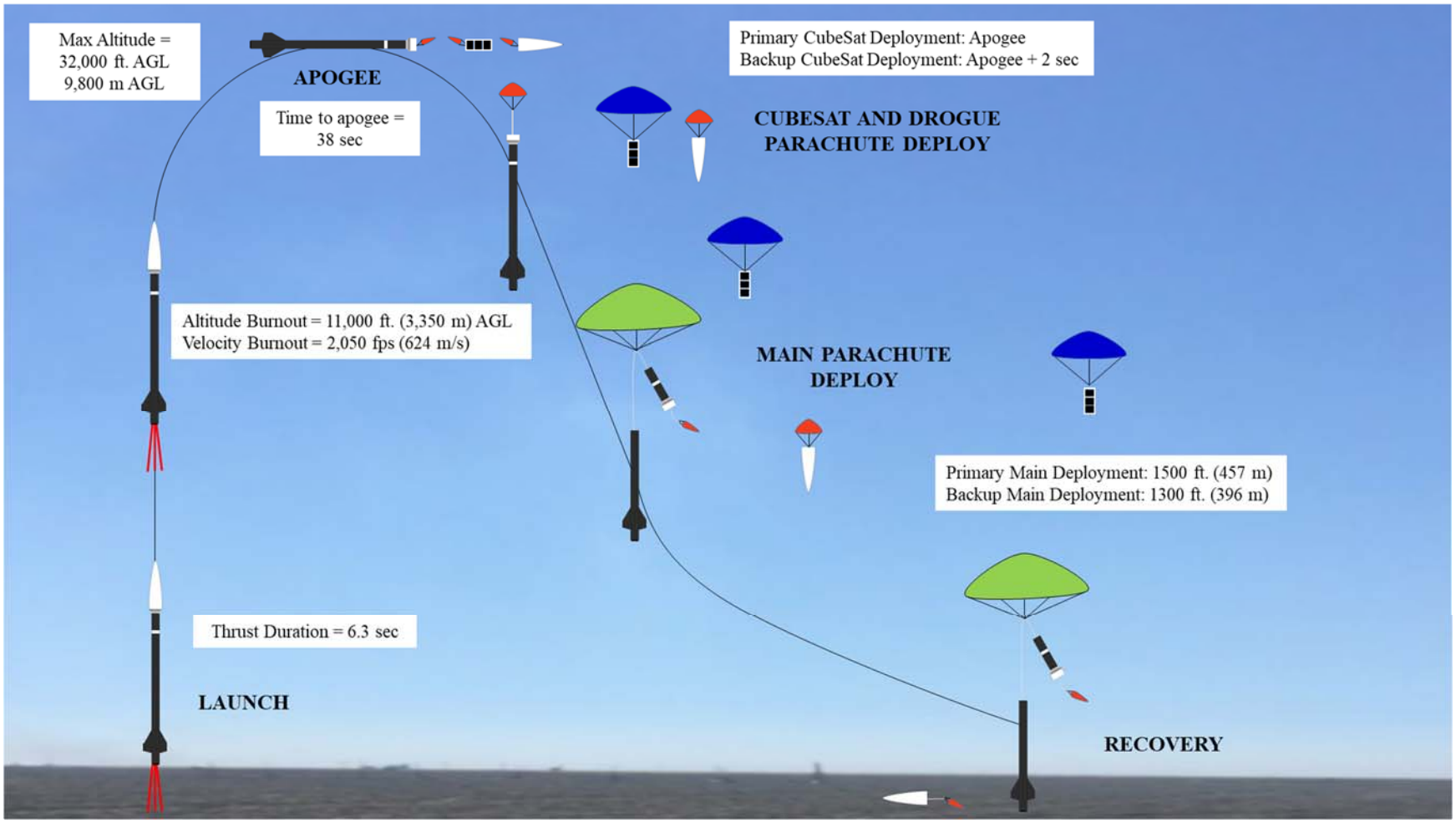


Figure 28. Planned Flight Profile of Sustainer Launch

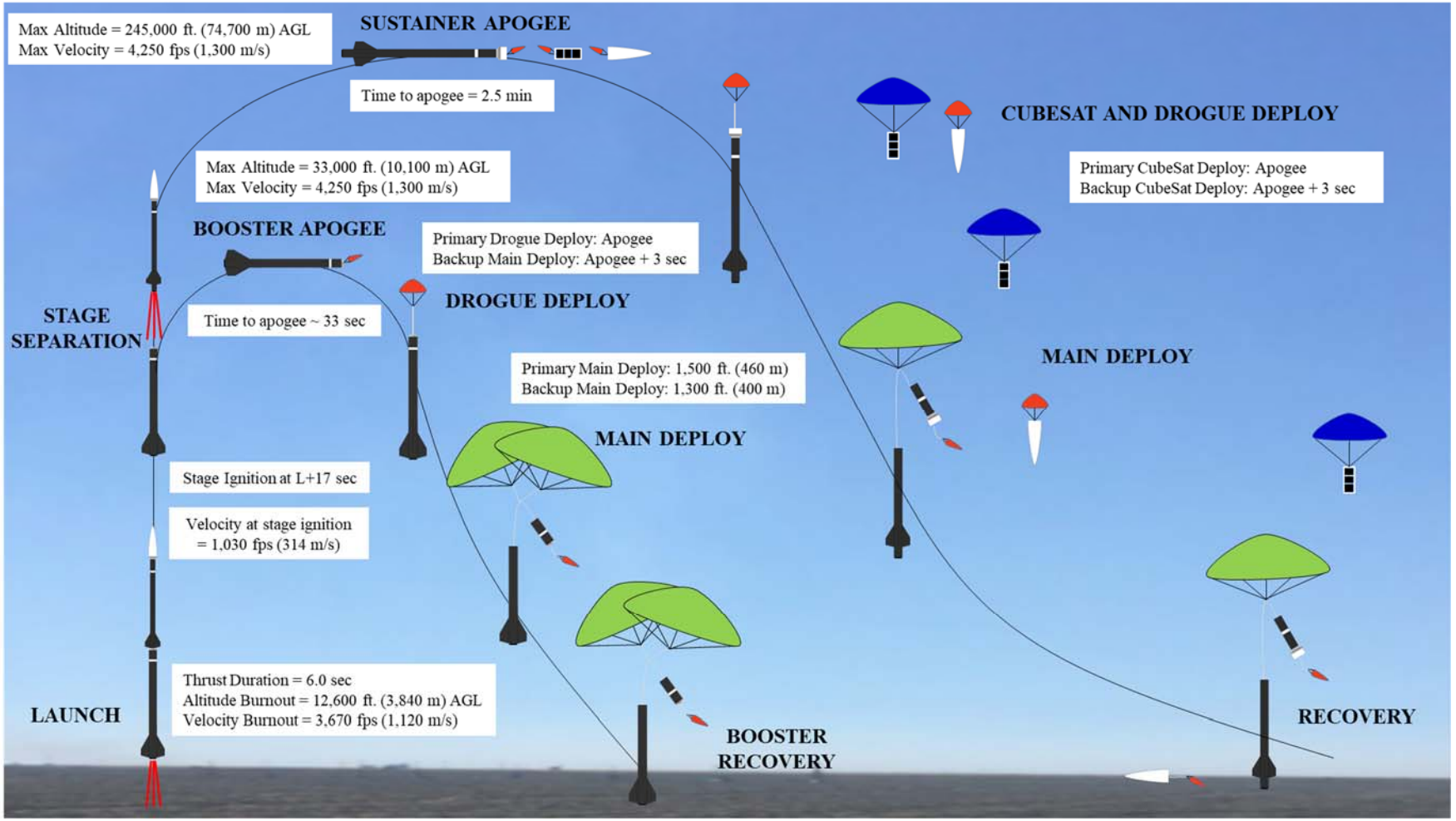


Figure 29. Planned Flight Profile for Full Mission Flight

C. STABILITY ANALYSIS

The NPS SSAG high-power rocket contains no active stabilization mechanisms. Instead, both sections of the rocket rely on restoring forces generated by the rocket's lift and drag force to achieve stable flight. To ensure rocket stability, the center of pressure (CP), the point through which the lift and drag forces act, must be aft of the center of gravity (CG), the point the rocket rotates around [57]. Figure 30 visually demonstrates this concept and shows how a rocket can passively achieve stable flight. In the diagram on the left, the CP is located aft of the CG. In this configuration, aerodynamic forces acting at the CP restore the rocket's orientation towards the direction of flight. In the diagram on the right, the CP is located forward of the CG. In this configuration, aerodynamic forces acting at the CP destabilize the rocket and force its orientation away from the direction of flight.

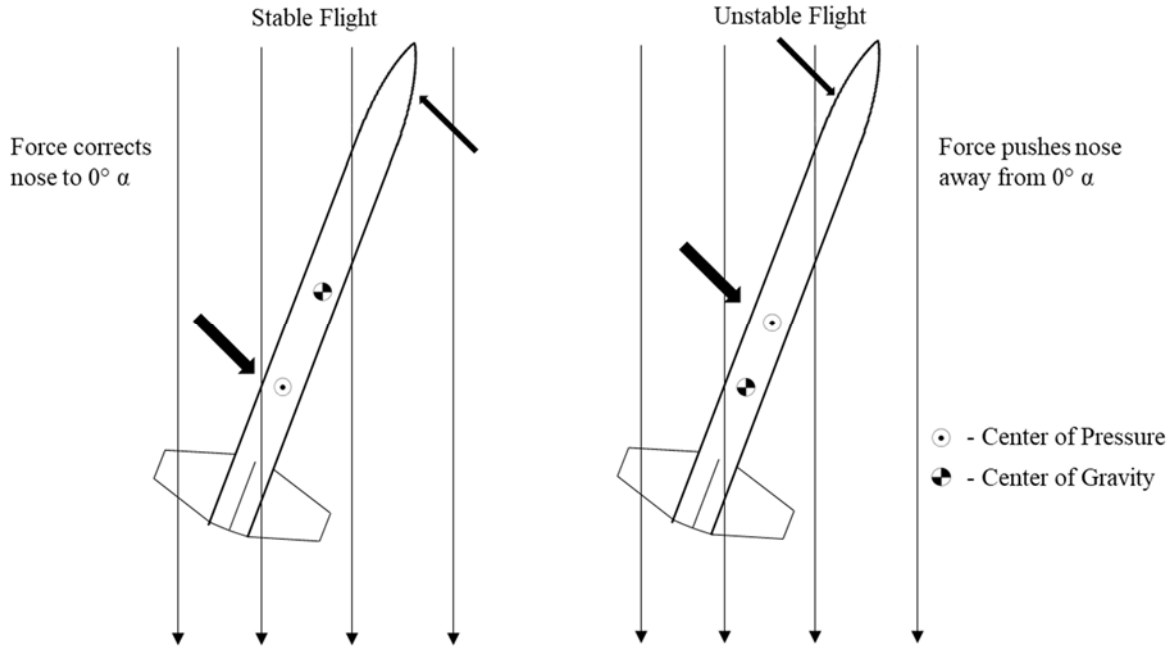


Figure 30. Rocket Stability Diagram

The location of the center of pressure in a rocket's design is primarily affected by the size and number of the fins, fin location on the rocket, and the size and length of the airframe. For the NPS SSAG high-power rocket, stability analysis of the initial design was necessary to ensure the rocket was sufficiently stable before commencing fabrication. Developed by James S. Barrowman in 1967, the Barrowman stability equations provide a practical solution for calculating the CP for slender-finned vehicles [58]. Stability analysis was conducted using the Barrowman Equations and RockSim. Appendix A provides the MATLAB script used to calculate the NPS SSAG high-power rocket CP using the Barrowman method. Results from the Barrowman and RockSim analysis are displayed in Table 11 and Figure 31. Examination of both the table and diagram indicate the rocket is sufficiently stable in both the single and double stage configuration.

Table 11. Stability Analysis Results for NPS SSAG High-Power Rocket Single Stage and Full Mission Configurations

Value	RockSim Result (From Nose Cone Tip)	Margin[1]	Barrowman Equation Result (From Nose Cone Tip)	Margin
Sustainer (4"-10 cm diam.)				
Center of Gravity	85 in 2.2 m	-	-	-
Center of Pressure	97 in 2.5 m	1.89	89 in 2.3 m	0.62
Sustainer and Booster (6"-15 cm diam.)				
Center of Gravity	172 in 4.4 m	-	-	-
Center of Pressure	189 in 4.8 m	2.76	182 in 4.6 m	1.63

[1] Margin is a dimensionless quantity calculated by taking the difference in location of the CP and CG and dividing by the diameter of the rocket. A margin of at least 1 is desired for passive stability to account for approximations in stability calculations.

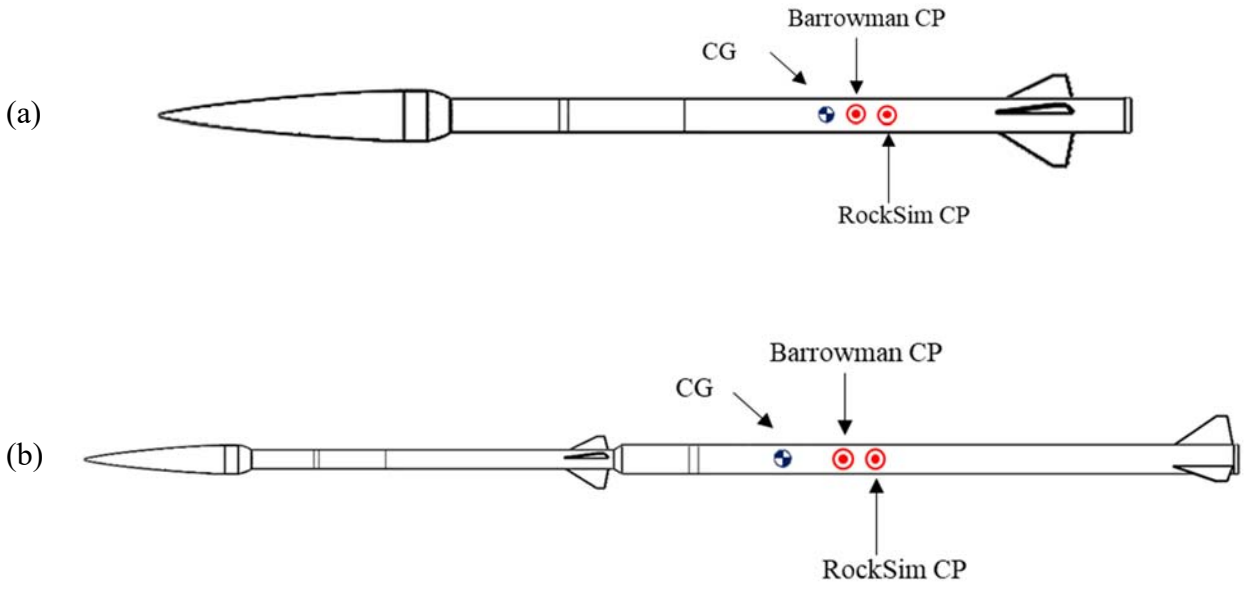


Figure 31. Results of Stability Analysis for (a) the Sustainer and (b) the Combined Sustainer and Booster

D. ROCKET COMPONENTS

This section of the chapter details the various subcomponents of the rocket. The section begins with the large structural components and moves inside the rocket to the various electronics.

1. Airframe

The airframe of both sections of the rocket is made of 0.072 in. (1.8 mm) thick carbon fiber tubing produced by Public Missiles Ltd. Carbon fiber was selected as the airframe material due to its high strength-to-weight ratio. In addition, the unique convolute wrap technique used by Public Missiles in manufacturing their carbon fiber tubes increases the tubing hoop and column strength, increasing the rocket's resistance to buckling.

2. Nose Cone

The nose cone of the rocket is a 6 in. (152 mm) filament-wound fiberglass nose cone with a metal tip produced by Madcow Rocketry. Fiberglass was selected as the nose

cone material due to its RF transparent properties. The shape of the nose cone is the mathematically derived Von Kármán profile. The Von Kármán profile is the theoretical minimum drag profile for a given length and diameter [59]. It was selected for this particular design for its relatively low drag through the transonic region of flight and its commercial availability. The Von Kármán shape is determined via Equation 1, where R is the radius of the base, L is the length, and x is a variable from 0 to L [59].

$$y = \frac{R}{\sqrt{\pi}} \sqrt{\arccos\left(1 - \frac{2x}{L}\right) - \frac{\sin\left(2\arccos\left(1 - \frac{2x}{L}\right)\right)}{2}} \tag{1}$$

Figure 32 is a plot of a notional nose cone with a Von Kármán profile. In the example the diameter of the nose cone is 6 in. (152 mm) and the length is 34.5 in. (87.6 cm).

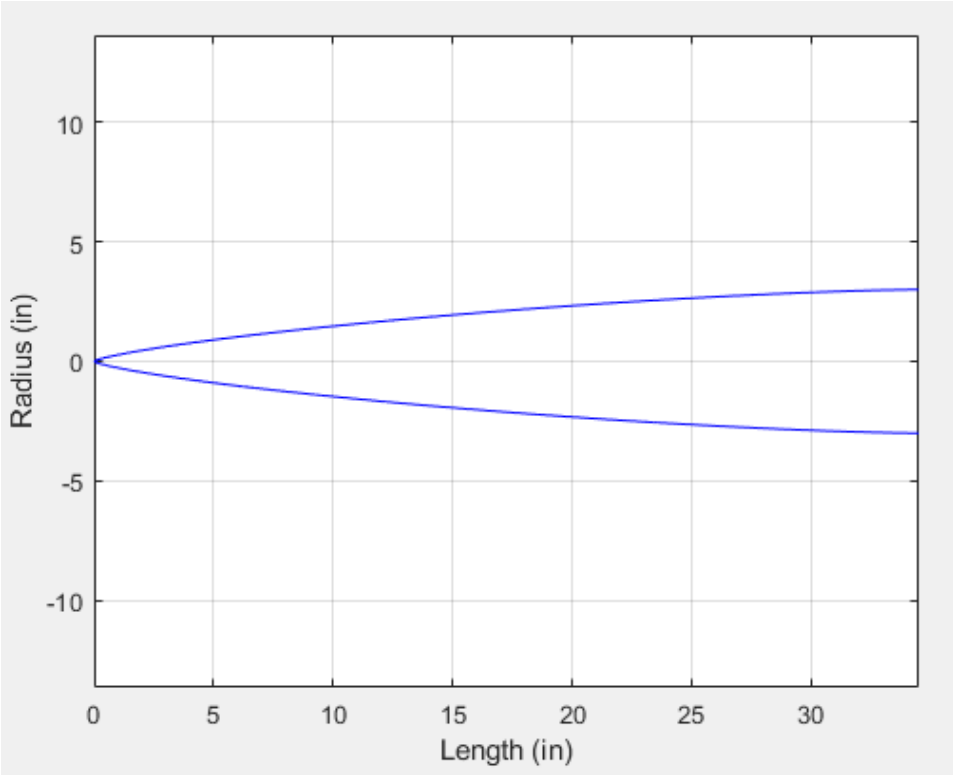


Figure 32. Plot of Von Kármán Profile

Within the nose cone, there is an electronics mounting and retention device. Figure 33 is a CAD rendering of the complete assembly. It shows how the assembly is composed within the nose cone and lists the various components of the device. In the upper left-hand portion of the image, the forward retention ring is epoxied into the uppermost portion of the nose cone and vertically retains the entire device. The aluminum rod is threaded into the forward retention ring, and the electronics mounting plate is placed on the aluminum rod. The rear bulk plate, which is sized to match the profile of the Von Kármán nose cone, holds a Session GoPro for inflight video and is held in place by the steel eye nut. The nose cone's parachute is connected via a ¼ in. (0.64 cm) quick link and Kevlar shock cord to the steel eye nut.

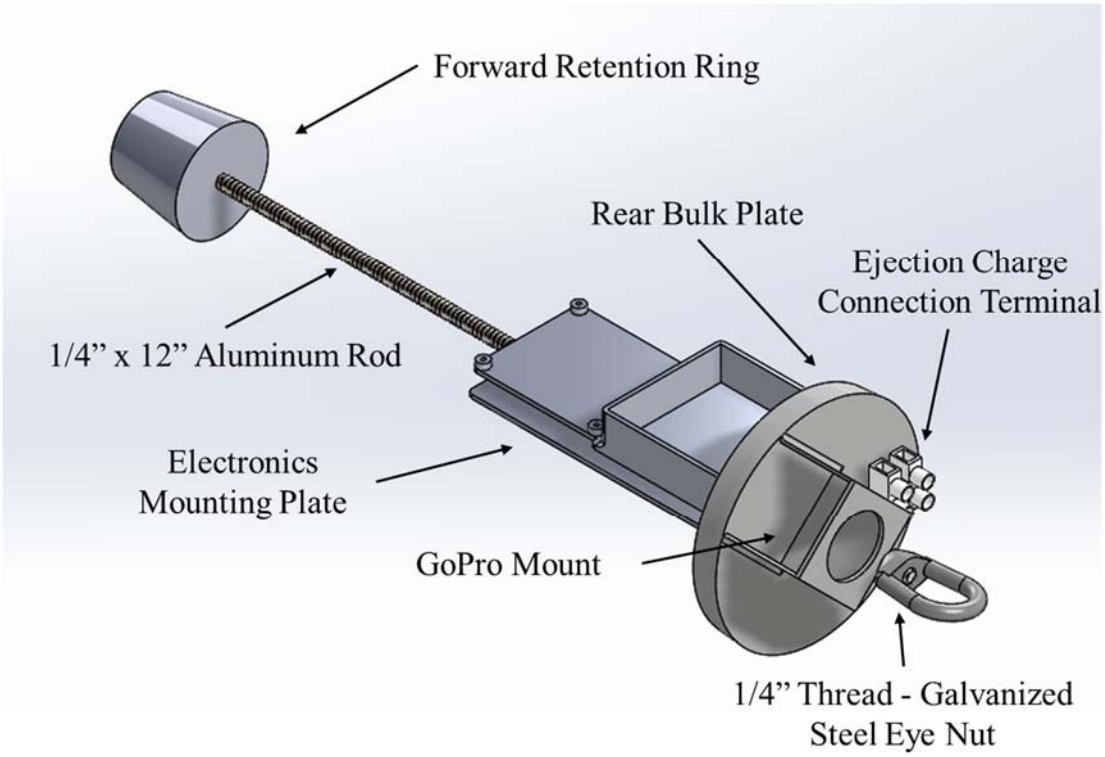


Figure 33. Upper Nose Cone Electronics Mounting Assembly (Nose Cone – Mount Assembly. SLDASM)

Figure 34 and Figure 35 provide an annotated top and bottom view of the 3D-printed electronics mounting sled that retained the nose cone electronic devices. Of note

in the image is the compact and narrow form-factor of the electronics mounting sled. The overall dimensions of the mounting sled are 2.5” (6.4 cm) W x 5” (13 cm) L x 2.5” (6.4 cm) H. To prevent rotation of the platform around the aluminum rod, the piece was sized to seat against the forward section of the nose cone.

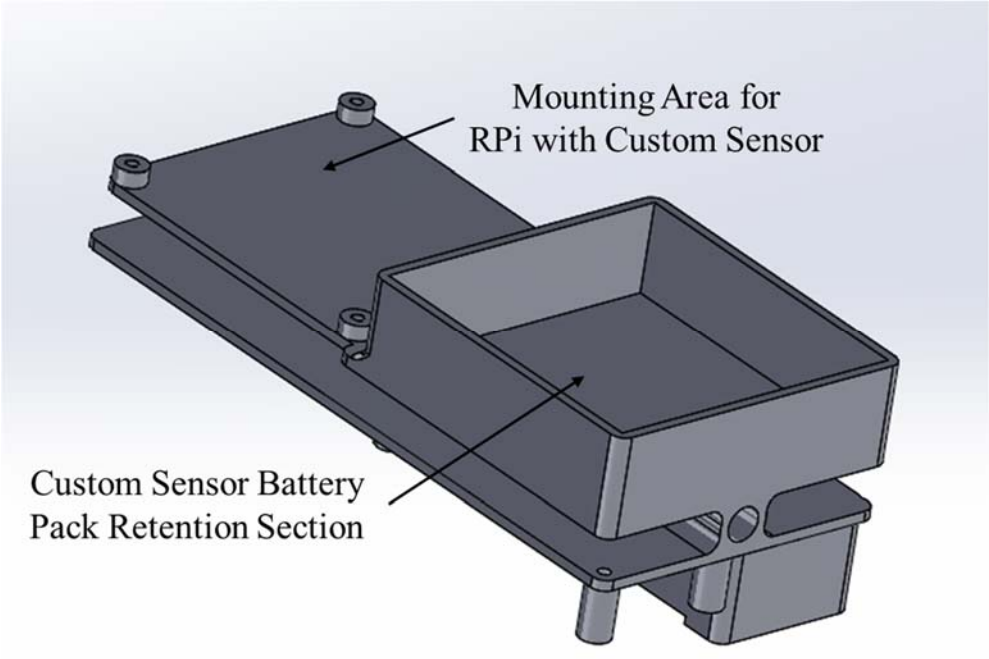


Figure 34. Nose Cone Electronics Mounting Sled (Top View)
(Nose Cone – Electronics Sled. SLDPRT)

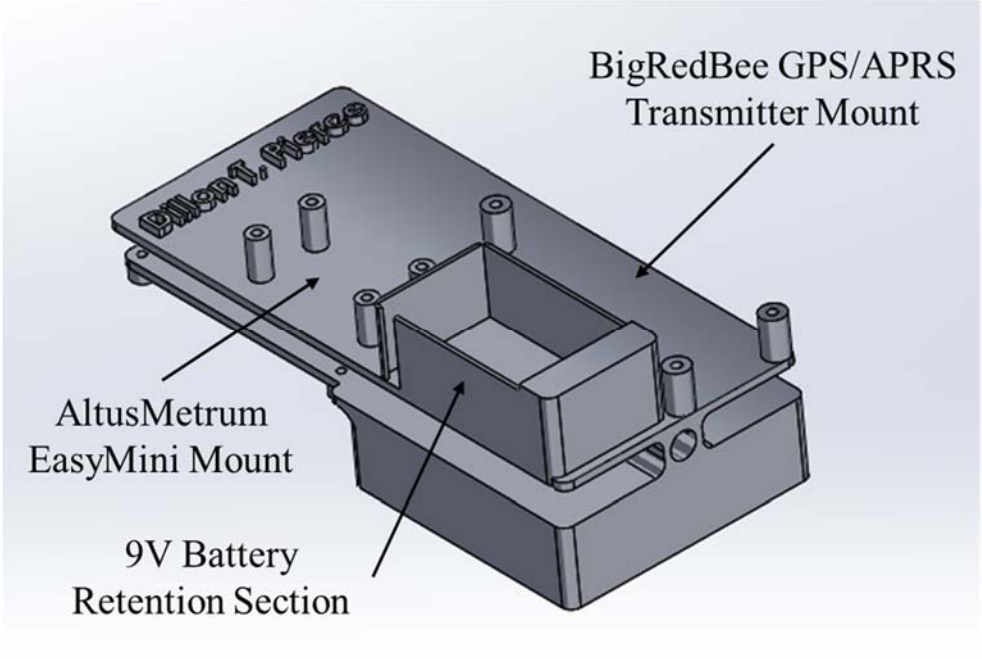


Figure 35. Nose Cone Electronics Mounting Sled (Bottom View) (Nose Cone – Electronics Sled. SLDPRT)

3. Recovery System

The recovery system of the rocket consists of the rocket's parachutes and parachute rigging hardware. Proper dimensions for recovery hardware in terms of parachute size and rigging length are vital to the safe landing of the rocket and enables rocket reuse.

Fruity Chutes was selected as the parachute of choice for this project due to their exceptional quality and flight history [60]. Kevlar, as opposed to nylon or cotton, was selected as the parachute shock cord material for its high strength and heat-resistive properties due to lessons learned in Chapter III Section B 2.2. Each section of the recovery hardware is connected via ¼ in. (0.64 cm) steel quick links for convenience purposes. Table 12 lists the parachute and shock-cord configurations for each section of the rocket.

Table 12. Sectional Parachute and Shock Cord Configurations

Section	Drogue Parachute	Main Parachute	Shock Cord
Nose Cone	-	48” (1.2 m) Fruity Chute: Classic Elliptical	Kevlar – 3 ft. (0.91 m)
Sustainer	24” (0.61 m) Fruity Chute: Classic Elliptical	84” (2.1 m) Fruity Chute: Iris Ultra Parachute	Kevlar – 30 ft. (9.1 m)
Booster	36” (0.91 m) Fruity Chute: Classic Elliptical	2 x 84” (2.1 m) Fruity Chute: Iris Ultra Parachute	Kevlar – 30 ft. (9.1 m)

Parachute sizes were determined by calculating each section’s descent rate based on the rocket’s estimated weight. Appendix A contains the MATLAB script named parachute descent rate calculator that was used for these calculations. The target descent rate for each section under the drogue parachute is ≤70 fps (21 m/s). The target descent rate for each section under the main parachute is ≤15 fps (4.6 m/s).

The recovery configuration dimensions for both the sustainer and booster portions of the rocket are depicted in Figure 36 and Figure 37, respectively. Of note in the image are the relatively long shock cord lengths connecting the different sections of the rocket. The configuration was specifically designed in this fashion for considerations concerning the energy dissipation of Kevlar shock cords due to the lack of stretch associated with the material. Longer shock cord lengths allow for some of the energy associated with section separation to be dissipated prior to the Kevlar cord reaching its maximum extent. Thus, the generated forces on the cord are less and potential mechanical failures are mitigated.

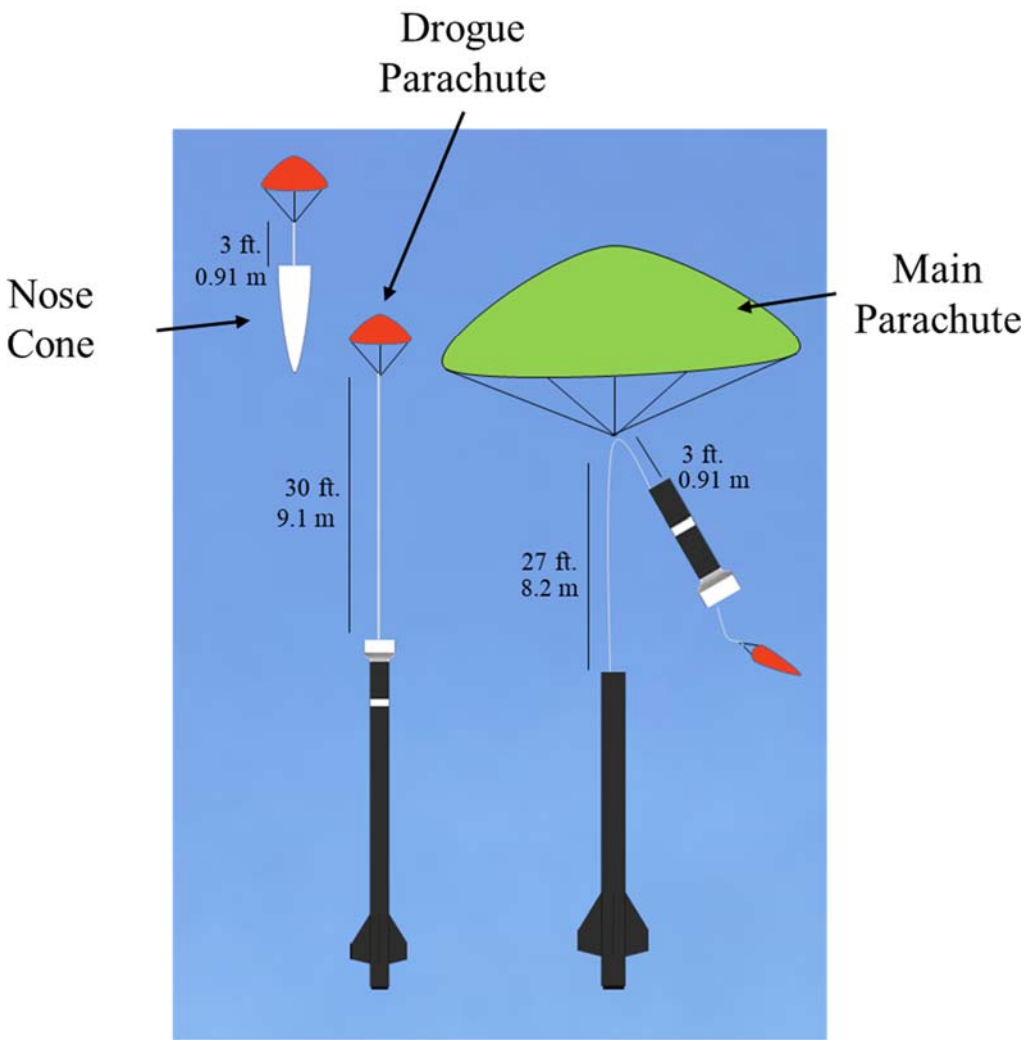


Figure 36. Sustainer Recovery Configuration

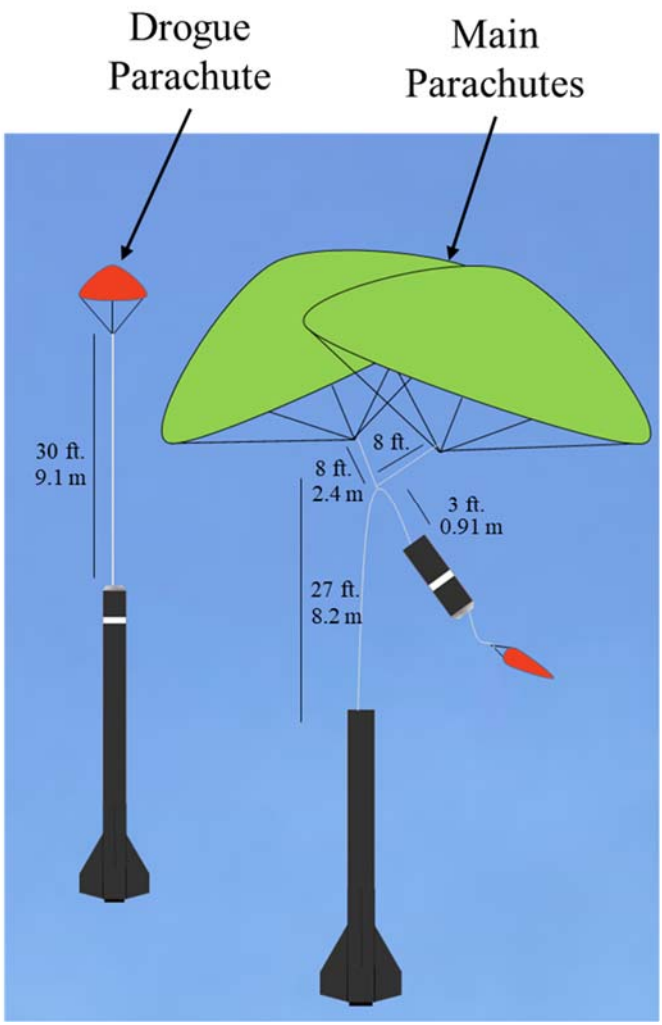


Figure 37. Booster Recovery Configuration

Figure 38 displays an example output plot from the MATLAB script named parachute descent rate calculator used for sustainer recovery analysis. Of note in the image is the initial rapid descent of the sustainer stage due to the low air density. At 1,500 ft. (460 m), the main parachute is deployed, and the sustainer slows to its target descent rate for landing.

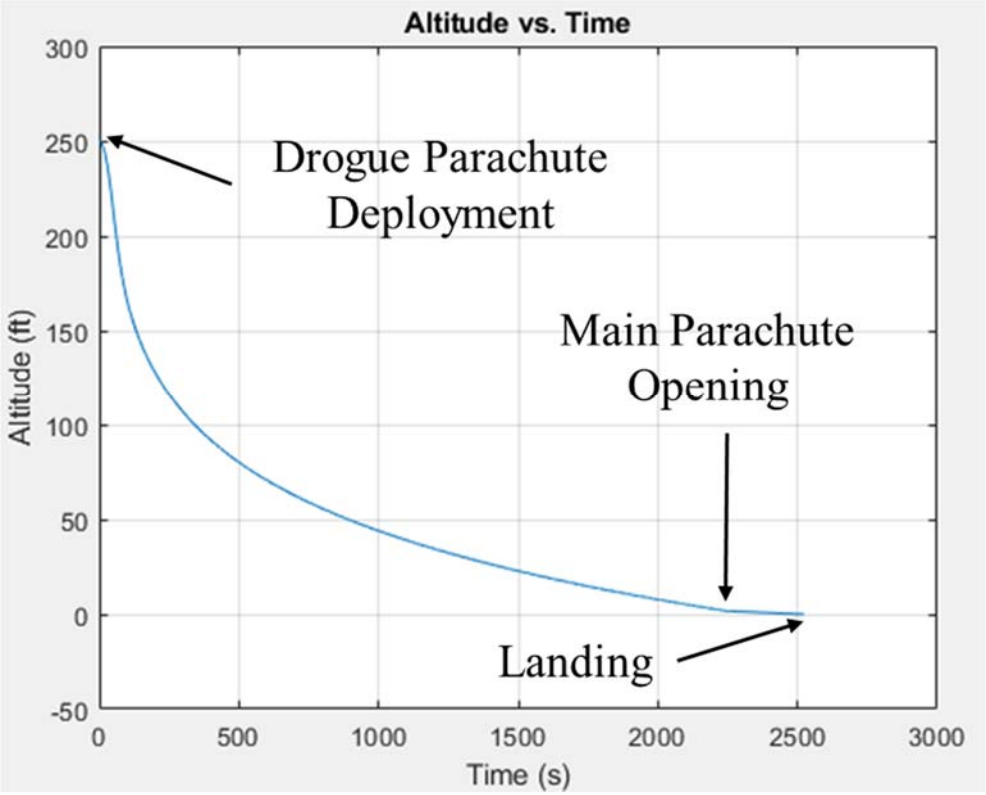


Figure 38. Sustainer Recovery Flight Profile

Table 14 lists the results of the author's MATLAB analysis for each section of the rocket. The inputs to the program, listed in Table 13, are the maximum projected altitude, the diameters and coefficients of drag of the drogue and main parachutes, and the mass of the rocket or section. Examination of Table 14 indicates that each section of the rocket is within the desired descent rate range of ≤70 fps (21 m/s) during main parachute opening and ≤15 fps (4.5 m/s) during landing.

Table 13. Section Recovery Dimensions

Section	Mass	Maximum Altitude - AGL	Drogue C_d	Main C_d	Drogue Diameter	Main Diameter
Nose Cone	4 lb. 1.8 kg	250,000 ft. 76,000 m	-	2.2	-	42 in 1.1 m
Sustainer	20 lb. 9.1kg	250,000 ft. 76,000 m	2	2.2	24 in 0.61 m	84 in 2.1 m
Booster	45 lb. 20.4 kg	33,000 ft. 10,000 m	2.2	2.2	36 in 0.91 m	2 x 84 in 2 x 2.1 m

Table 14. Section Recovery Descent Rates and Time

Section	Descent Rate at Main Parachute Opening	Descent Rate at Landing	Total Elapsed Time Under Canopy
Nose Cone	-	13 fps 4.0 m/s	66 min
Sustainer	62 fps 19 m/s	14 fps 4.3 m/s	16 min
Booster	65 fps 20 m/s	11 fps 3.4 m/s	8.8 min

4. Forward Transition

The forward transition of the rocket serves to connect the larger diameter nose cone to the smaller diameter upper sustainer airframe. It is a custom design and made of 7075 aluminum alloy. Figure 39 displays a sectional CAD rendering of the forward transition with annotated overall dimensions. Of note in the image is that the transition is connected to the nose cone via two 2/56 nylon shear pins and to the sustainer upper airframe via four 6–32 stainless steel screws. Additionally, the pseudo-foil shape of the 3 in. (7.6 cm) transition region between the 1 in. (2.5 cm) shoulders was intended to reduce pressure drag on the rocket.

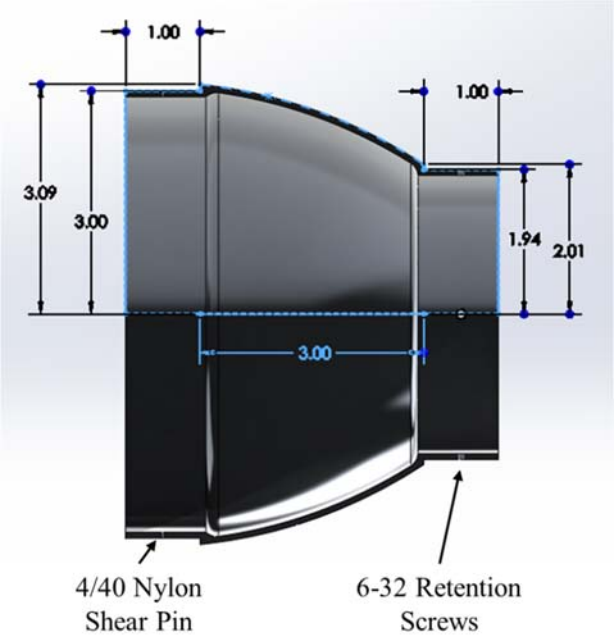


Figure 39. CAD Rendering of Forward Transition (Transition – New.SLDPRT)

5. Rear Transition

The rear transition of the rocket serves to connect the smaller-diameter sustainer to the larger-diameter booster. It is a custom design and made of 7075 aluminum alloy. Figure 40 displays a sectional CAD rendering of the rear transition piece with annotated overall dimensions. For stability, the sustainer motor is designed to slide approximately 7 in. (18 cm) into the rear transition and seat against the rear wall pictured in the figure. The configuration is designed to transfer the forces generated by aerodynamic pressure on the sustainer through the rear transition and to the body of the booster airframe.

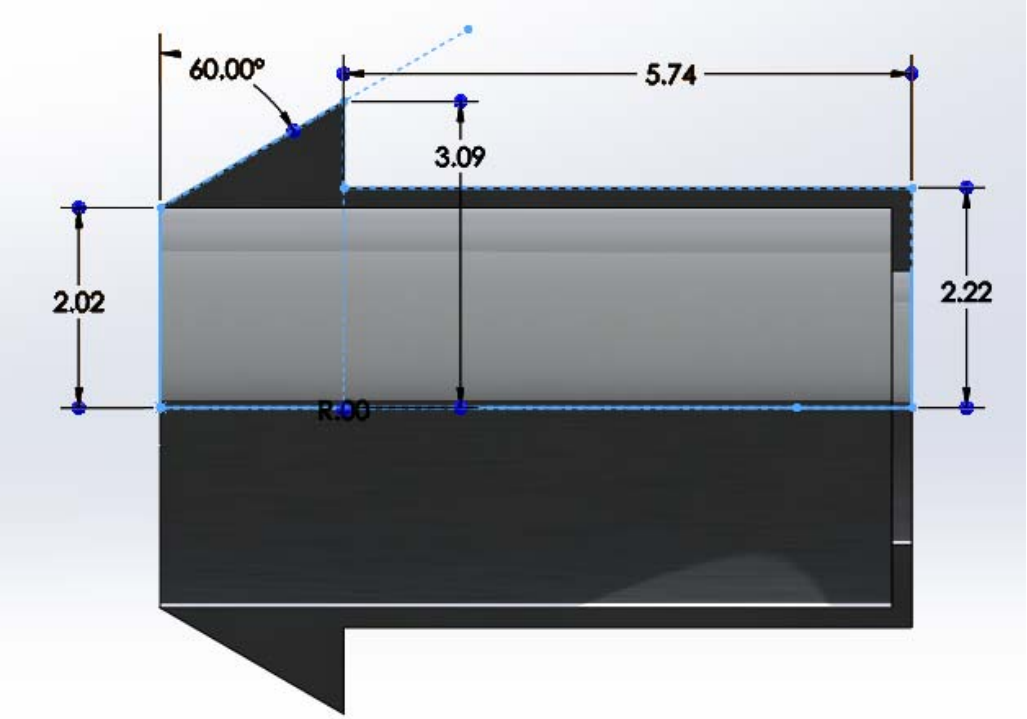


Figure 40. Sectional CAD Rendering of Rear Transition (Transition – Sustainer to Booster.SLDPRT)

Staging is controlled through the rear transition piece by the booster flight computers. This configuration reduces the chances of optimizing the timing of the sustainer motor ignition, as the sustainer must be lit before the booster detaches and thereby reducing sustainer coast time.[4] But, it is simpler than controlling staging from the sustainer stage as the igniter leads can be inserted through the aft end of the motor. Figure 41 depicts a CAD rendering of the staging configuration. In the image, the staging igniter leads can be seen proceeding through the aft end of the sustainer motor. The direct route simplifies the wiring and makes for a more robust staging design.

[4] Because the ignition of the sustainer motor is controlled by the booster electronics, sustainer motor ignition must occur before the two stages separate. If by contrast the sustainer electronics controlled the sustainer motor ignition, the sustainer section could coast longer without the booster before igniting the motor. Thus altitude could be optimized by the longer coast time and delayed sustainer motor ignition.

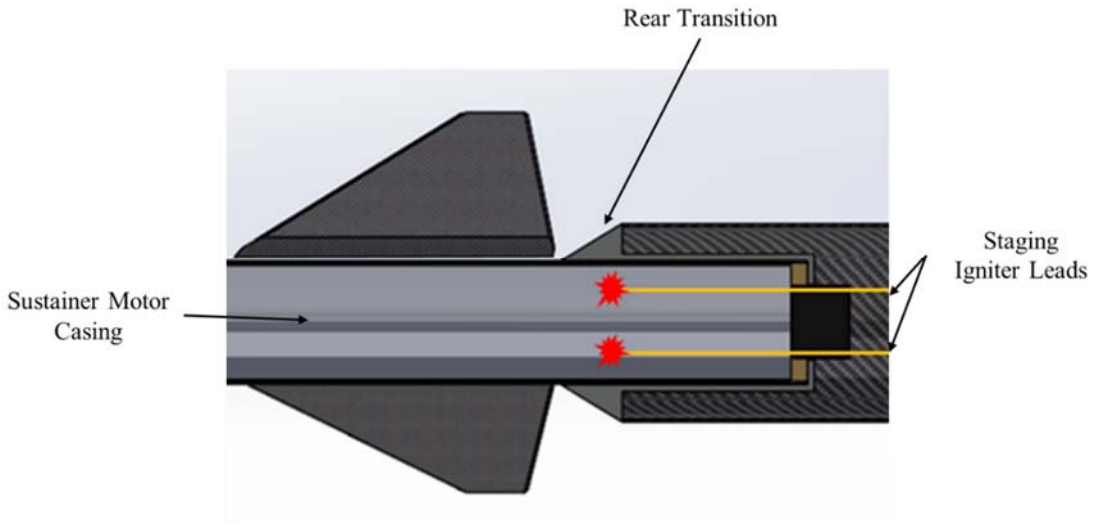


Figure 41. CAD Rendering of Staging Configuration

6. Fins

The fins for both the booster and sustainer sections of the rocket are made of 1/4”-(6.4 mm) thick carbon fiber sheeting with a birch wood core and were produced by Dragon Plate. According to Dragon Plate, the “composite sandwich combines the superior strength and stiffness properties of carbon-fiber with a lower density core material” and “[creates] a final product with a much higher stiffness to weight ratio than with either alone” [61]. Figure 42 depicts a CAD rendering of both the sustainer and booster fins and displays the dimensions of each. The relatively aggressive fin sweep angle is designed to reduce aerodynamic drag and fin loading during transonic flight. Other dimensions such as cord, tip, and sweep length were generated during the stability analysis of each section.

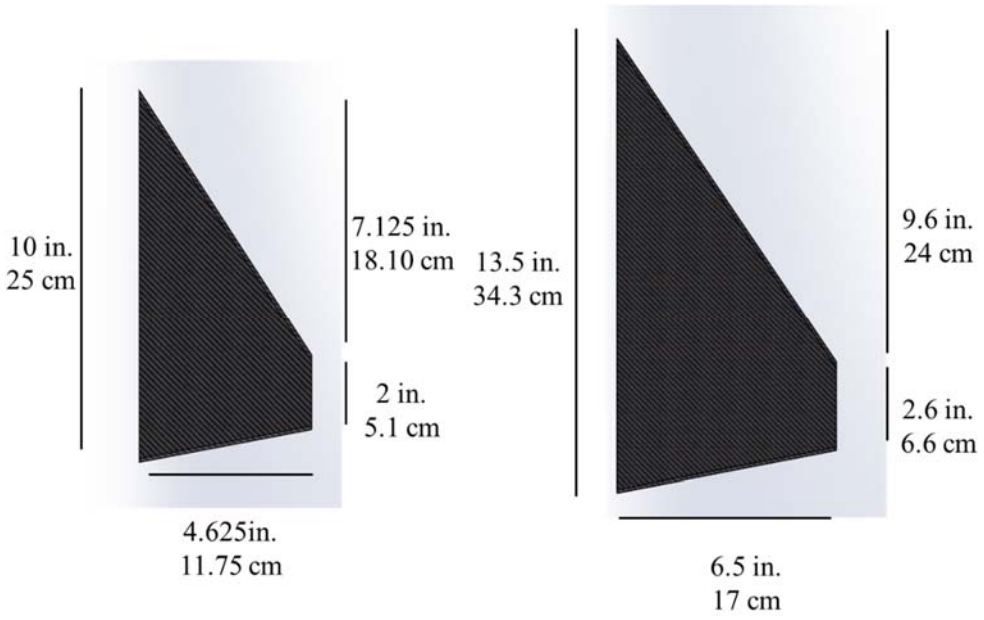


Figure 42. Sustainer (left) and Booster (right) Fin Dimensions

The fins are aligned and tacked on to the airframe using a custom-designed 3D-printed fin alignment guide. Figure 43 is a picture of the sustainer section of the airframe during the attachment of the sustainer fins. The fin alignment guide depicted in the image fits snugly over the rocket's airframe and holds the tip of each fin in place while the epoxy cures. Due to the minimum-diameter design of each section, the fins have minimal contact with the rocket's airframe. To reinforce the fin–airframe joint, epoxy fillets are added. Figure 44 depicts the epoxy fillets that were added to the sustainer fin-airframe joint. Further reinforcement of the fin attachment point is accomplished through the application of an additional layer of 3k 8harness satin weave carbon fiber cloth, which connects the various fin segments. The layer is epoxied in place and affixed to the rocket using a vacuum-bagging technique. Figure 45 depicts the sustainer fin can after the vacuum bag process and prior to sanding. It can be noted from the image that there is an increase in contact surface area between the fin and airframe with the addition of the carbon fiber cloth. Such an overlap increases the lateral strength of the fin–airframe joint and mitigates fin flutter through the transonic region of flight.

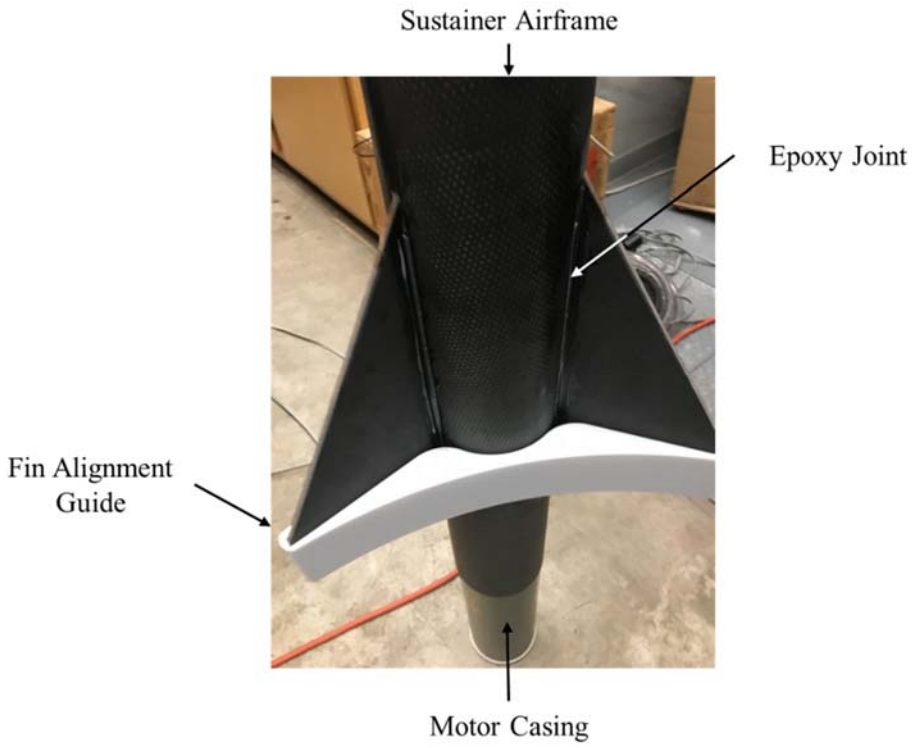


Figure 43. Sustainer Fin Construction

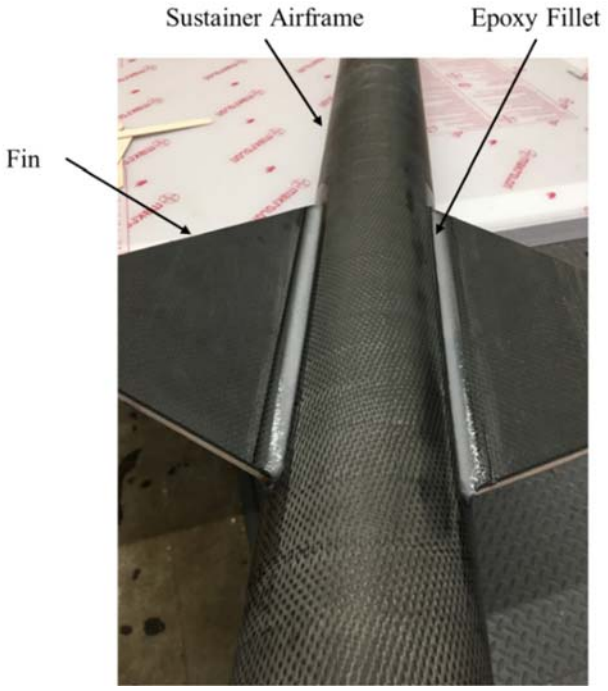


Figure 44. Sustainer Fin Epoxy Fillets

Figure 45. Sustainer Fin Can with Added Carbon Fiber Layer

7. Electronics Bay

The electronics bay of the rocket is designed to house the flight computers and serve as a mount for the CO_2 and black powder ejection systems. Each section of the rocket has an independent electronics bay from which all flight events are controlled. For the NPS SSAG high-power rocket, each electronics bay is made of G12 filament-wound fiberglass produced by Madcow Rocketry. Fiberglass was selected as the material of choice for its RF-transparent properties which permit flight computer telemetry from the rocket. Figure 46 and Figure 47 depict a CAD rendering of the electronics bay designed for each section of the rocket.

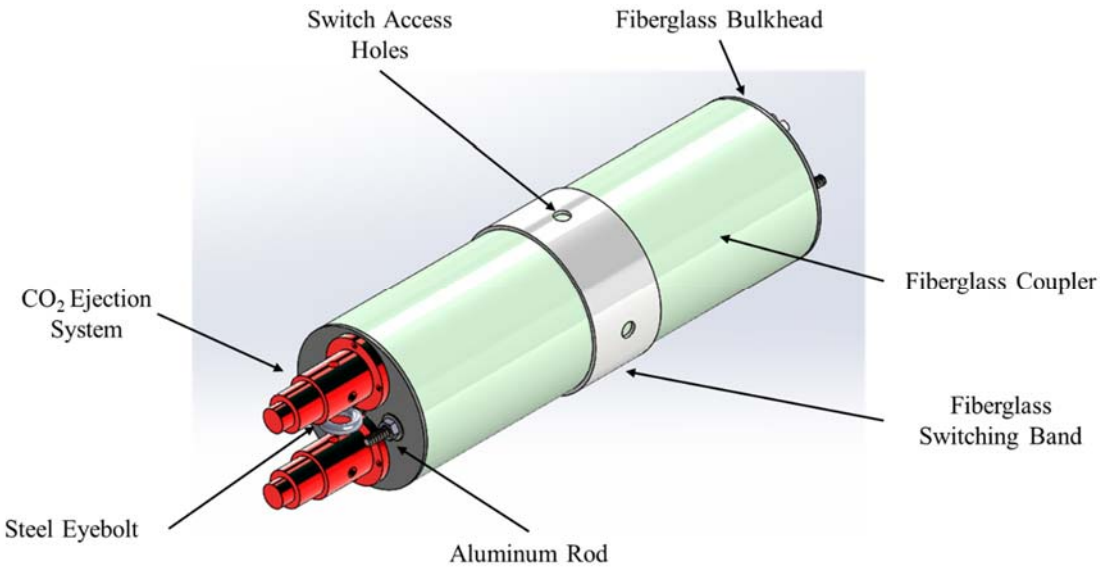


Figure 46. CAD Rendering of Electronics Bay (Electronics Bay.SLDASM)

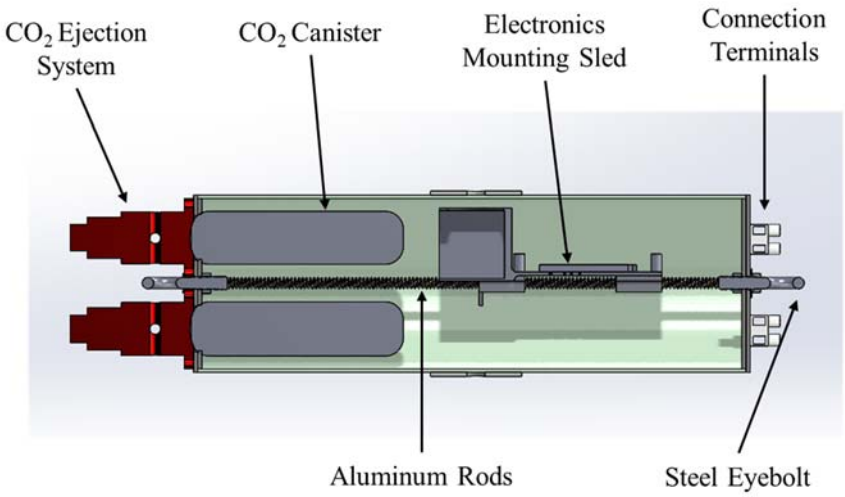


Figure 47. Cutaway CAD Rendering of Electronics Bay (Electronics Bay.SLDASM)

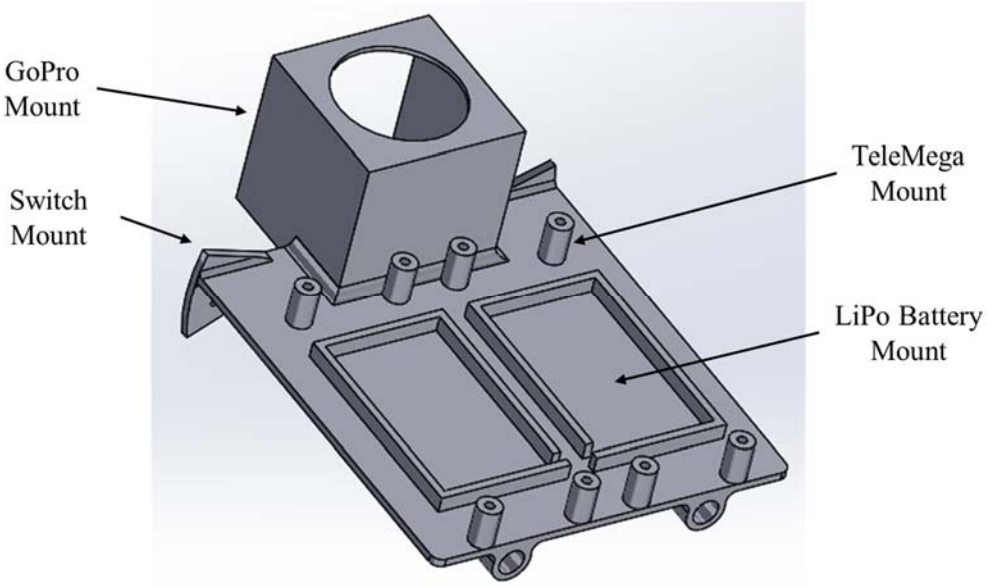


Figure 48. Electronics Mounting Sled (Sustainer E-Bay Sled.SLDPRT)

8. Ejection System

The parachute and payload ejection system is composed of both black powder charges and CO_2 canisters. For the drogue parachute and payload ejection charges, which need to function at high altitudes, CO_2 systems are used for the reasons stated in the previous lessons learned (Chapter III Section D 2.2). The CO_2 ejection system is known as the RAPTOR and is produced by Tinder Rocketry. The RAPTOR system was selected for its ease of installation and robust design. The system functions by using a small, tightly sealed black powder charge to puncture a CO_2 canister. The rapidly expanding CO_2 generates the necessary pressure to shear the shear pins and separate sections of the rocket. Subsequently, the parachutes and payload are deployed. Figure 49 shows an expanded CAD rendering of the RAPTOR CO_2 ejection system. In the image, the charge cup contains the small amount of black powder. Once ignited, the pressure from the black powder charge forces the puncture piston to compress the return spring and puncture the CO_2 canister. The gas from the canister expands through the holes in the mounting cap and the parachutes and or payload are deployed.

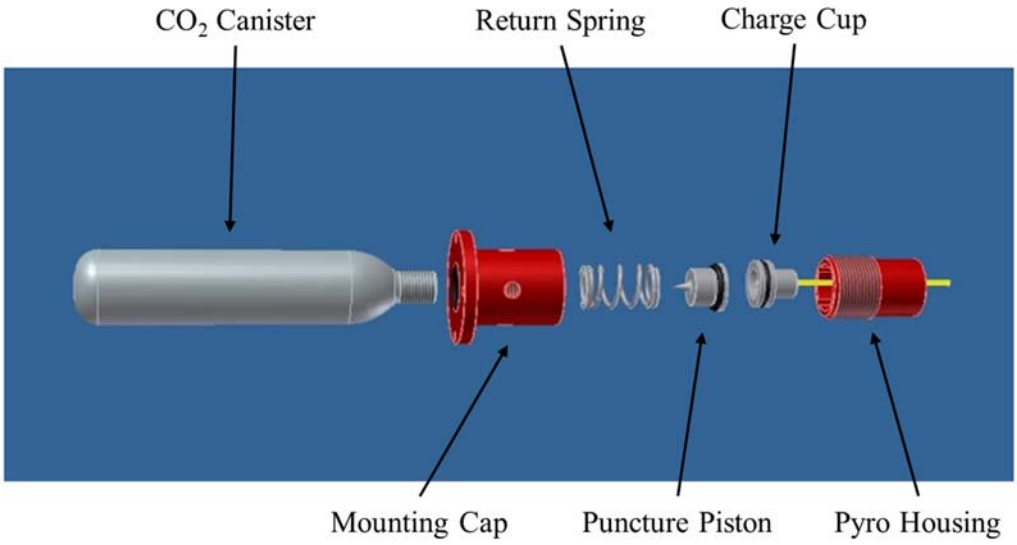


Figure 49. RAPTOR CO2 Ejection System. Adapted from [62].

The lower-altitude main-parachute charges contain only black powder. The black powder selected for use is GOEX FFFFg rifle black powder. FFFFg or "4Fg" refers to the black-powder grade and grain size. FFFFg black powder is sporting grade (as opposed to blasting grade) and has a grain size of 0.017 – 0.0060 in. (0.42 – 0.15 mm) [63]. It is the second smallest commercially available black powder grain size. The relatively small grain size increases the black powder burn rate which produces a more rapid buildup of pressure during ejection, making it desirable for the parachute ejection system. The black powder is contained in and ignited by a filament-style ejection canister made by Pratt Hobbies. Figure 50 is a simplified rendering of the filament style ejection canister. When initiated by the flight computers, the filament in the ejection canister ignites the black powder in the canister, and the resulting pressure separates the rocket sections and ejects the main parachute.

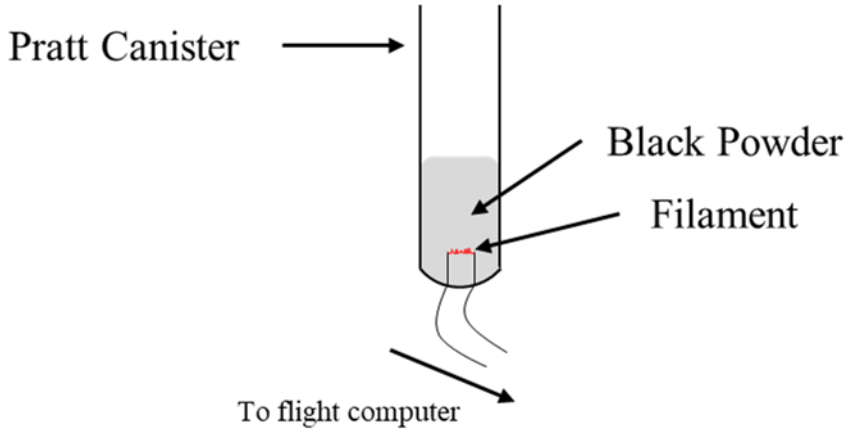


Figure 50. Filament Style Ejection Canister

An initial estimation for black-powder ejection-charge sizing was determined using a manipulation of the ideal gas law. The simplification of the equation is shown in Equation 2 where L is the length of the compartment containing the charge, r is the inside radius of that compartment, and pressure is the desired pressure in psi produced by the black powder charge. Temperature (1840 K) is the black powder combustion temperature found in [64]. Pressure can be determined using Equation 3 where the bulkhead force is

the pound-force required to shear the section's shear pins and SA is the surface area of the inner compartment bulkhead. The bulkhead force can be calculated by multiplying the number of shear pins in a respective section by the average shear strength of the pin. The 4–40 nylon screws that are used as the shear pins for the NPS SSAG high-power rocket have an average shear strength of 63 lbf (280 N) [65].

$$BP(grams) = \frac{pressure\left[\frac{lbf}{in^2}\right] * \pi\left(r[in]\right)^2 L[in]}{0.674\left[\frac{lbf \cdot in}{g \cdot K}\right] * 1840K} \tag{2}$$

$$pressure = \frac{Force_{Bulkhead}}{SA_{Bulkhead}} \tag{3}$$

As suggested by the RAPTOR ejection system documentation, the method for sizing CO_2 cartridges is to determine the amount of black powder required and multiply the quantity by a factor of five [66]. Table 15 summarizes the ejection charge sizes for each section of the rocket.

Table 15. Ejection Charge Sizing

Rocket Section	Radius (in) (cm)	Length (in) (cm)	Volume (in^3) (cm^3)	Shear Pins	Bulkhead Force (lbf) (N)	Est. BP Charge Size (lbm) (g)	Est. BP Charge 50% Margin (lbm) (g)	Estimated Size CO_2 (lbm) (g)
Nose Cone	Variable	34.5 87.6	560 9,200	2	126 561	0.004 2.0	0.0066 3.0	0.051[1] 23
Sustainer (Main Parachute)	2.0 5.1	18 46	226 3,700	3	189 841	0.0060 2.7	0.0090 4.1	-
Booster (Drogue Parachute)	3.0 7.6	4 10	113 1,850	3	189 841	0.0014 0.64	0.0021 0.96	0.051 23
Booster (Main Parachute)	3.0 7.6	15 38	424 6,950	3	189 841	0.0053 2.4	0.0079 3.6	-

[1]0.051 lbm (23 g) is the minimum CO_2 canister size

Ground testing of the payload ejection system was conducted to ensure the charges were sufficient for separating the nose cone and deploying the CubeSat payload. Figure 51 is a video capture from the test and shows the separation of the nose cone and subsequent CubeSat deployment. It was noted from these tests that the CO_2 ejection method was not as violent as the black powder charges. Therefore, significant margin was added to the CO_2 cartridge to ensure separation of the rocket's segments and CubeSat deployment while maintaining a low stress on the rocket and payload airframe. Instead of the 23g canister listed in Table 15, a 45g primary and a 75g backup CO_2 canister were used.

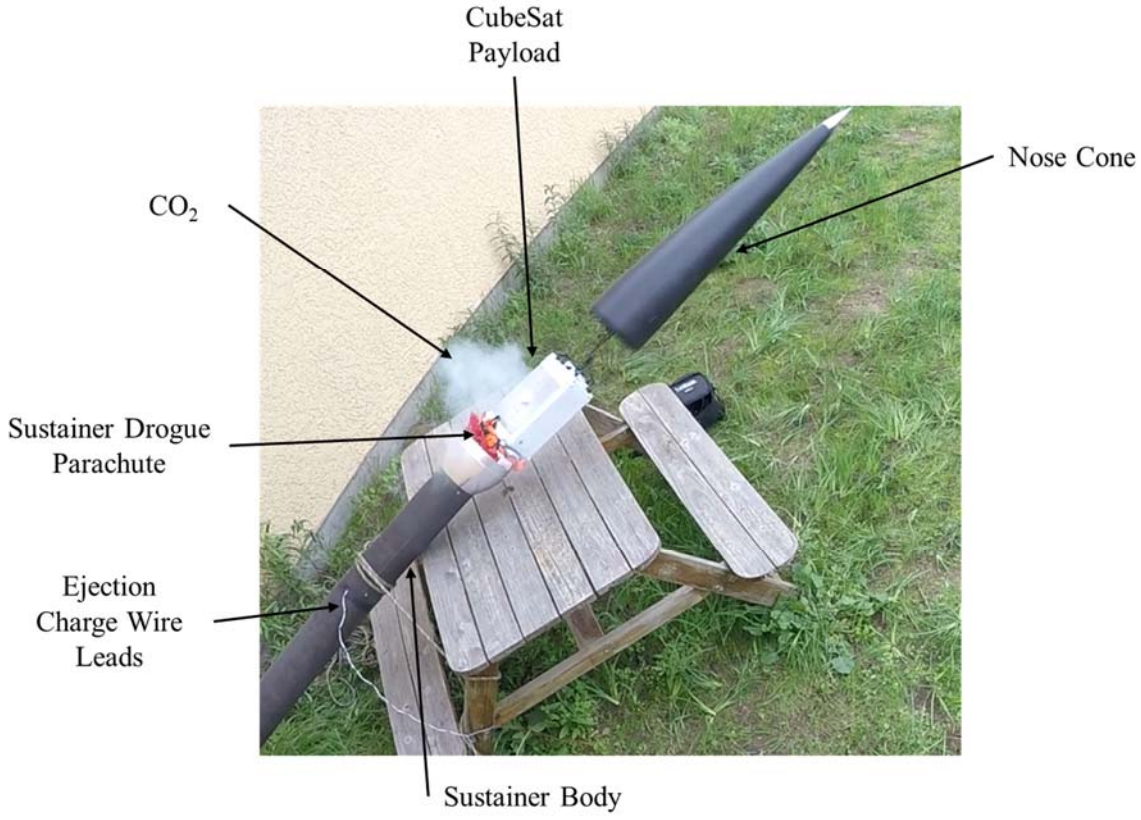


Figure 51. Payload Ejection Test

9. Electronics

The flight computers onboard the rocket control the various flight events such as payload and parachute deployment and section staging. In addition, flight sensors capture onboard data that aid in characterizing the rocket's flight and relaying the rocket's positional data. Table 16 lists the electronic devices carried by the rocket and the various details associated with their respective functionality, discussed in detail in the sections that follow.

Table 16. Rocket Electronic Devices

Device	Function	Sensors	Transmitter	Location
Altus Metrum TeleMega	General Data Logging Staging Parachute and Payload Ejection Stage Tracking	GPS High-G Accelerometer 3 Axis Accelerometer Gyroscope Magnetometer Altimeter	32mW 70cm amateur band	2 x Booster 2 x Sustainer
BigRedBee GPS/APRS Transmitter	GPS Tracking General Data Logging	GPS	100 mW 70cm amateur band	Nose Cone
RPi A+ with Custom Sensor Board	High Fidelity Data Logging GPS Tracking Telemetry	GPS High-G Accelerometer 3 Axis Accelerometer Gyroscope Magnetometer Altimeter	1W 915 MHz	Nose Cone

a. Altus Metrum TeleMega

The Altus Metrum TeleMega is a commercially available dual-deploy altimeter with integrated telemetry and positional tracking capabilities [67]. It is the main flight computer for each section of the rocket and controls all the flight events to include payload and parachute ejection and section staging. Both the booster and sustainer portions of the rocket contain redundant and independently powered systems for both

backup and safety purposes. Figure 52 displays the simplified wiring diagrams for the sustainer and booster portions of the rocket. In addition, Table 17 lists the various flight computer responsibilities and timing.

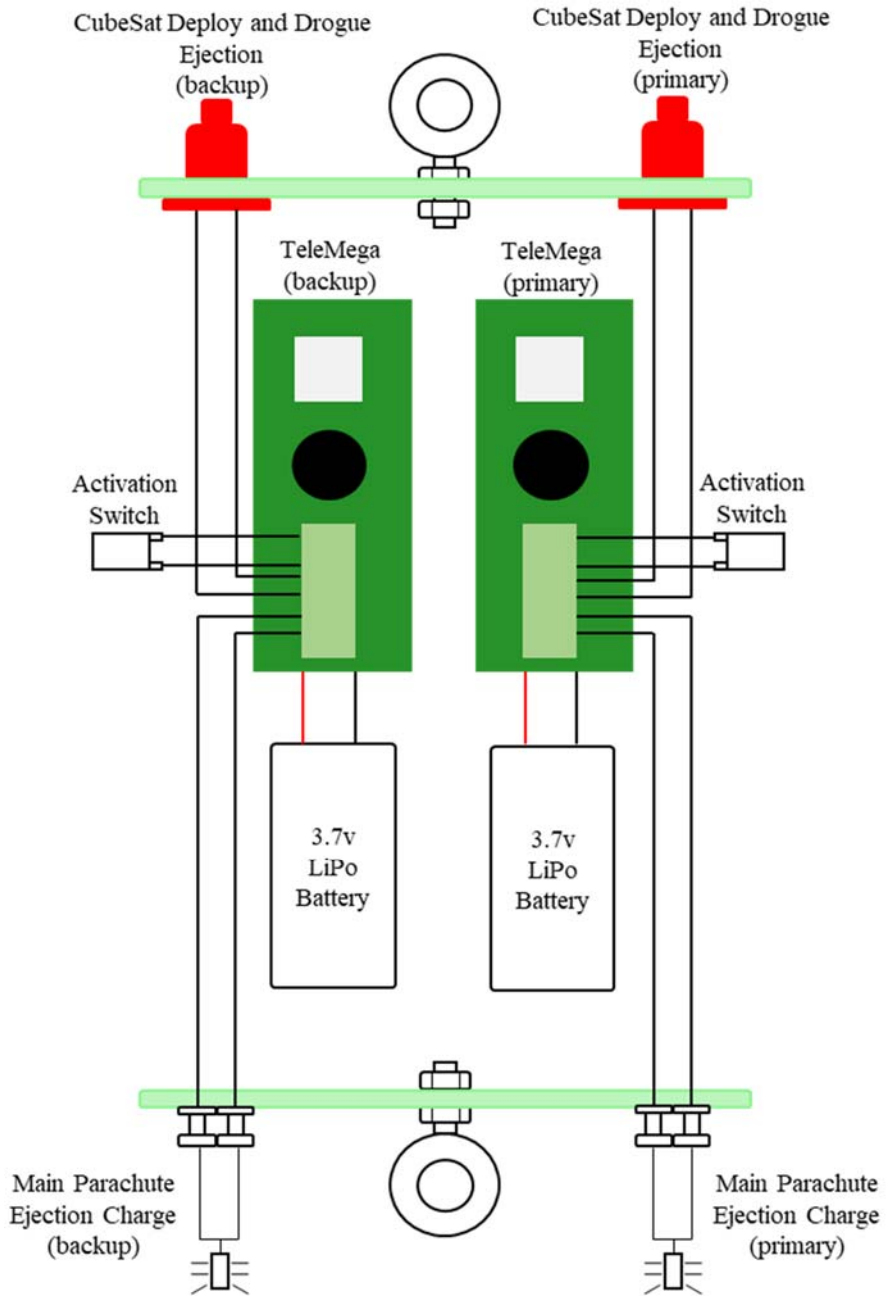


Figure 52. Flight Electronics Simplified Wiring Diagram

Table 17. Flight Computer Responsibilities and Timing

Device	Drogue Parachute	Main Parachute	Staging
Sustainer			
TeleMega (1)	Apogee (Primary)	1500 ft (460 m) (Primary)	
TeleMega (2)	Apogee + 2 sec (Backup)	1300 ft (400 m) (Backup)	
Booster			
TeleMega (3)	Apogee (Primary)	1500ft (460 m) (Primary)	L + 6 seconds
TeleMega (4)	Apogee + 2 sec (Backup)	1300ft (400 m) (Backup)	L + 6 seconds

Telemetry data from the TeleMega is received via the Altus Metrum TeleDongle, which serves as an RF interface for ground-station operations [68]. Both the TeleMega and TeleDongle are configured with Altus Metrum's AltOS firmware [69].

b. BigRedBee GPS/APRS Transmitter

The BigRedBee GPS/APRS Transmitter is a commercially available 100mW positional transmitter that operates on the 70cm amateur RF band [70]. In addition, the device has a non-volatile memory that is capable of storing over two hours of GPS data collected at a sampling frequency of 1 Hz [70]. For this thesis, APRS packets from the BigRedBee transmitter are received via a Kenwood TH-72A 440 MHz FM dual band radio. The Kenwood TH-72A radio was selected for receiving the BigRedBee telemetry due to its built-in programming, which supports APRS data formats, simplifying integration and operation with the BigRedBee transmitter [71].

c. Custom Sensor Board

The custom sensor board designed for the NPS high-power rocket runs on the RPi 3 A+ model as a standard dimension hardware attached on top (HAT). The eventual goal of the custom sensor board is to process all flight data, send all inflight telemetry, and control all of the rocket's flight events. However, for the purposes of this thesis, the sensor board functions as a data collection device and sends limited inflight telemetry (flown on flights 5 and 6). Details associated with the board's schematic, PCB design, and software are available in Appendices B and C. Figure 53 displays the custom sensor

board mounted on the RPi 3 A+ microcomputer. The larger sensors and components listed in the figure and some of the capabilities of each sensor are as follows:

- Ublox NEO M8N series GPS
- STMicroelectronics LSM9DS1 Inertial Measurement Unit (IMU)
  - ±2/±4/±8/±16 g linear acceleration full scale
  - ±4/±8/±12/±16 gauss magnetic full scale
  - ±245/±500/±2000 dps angular rate full scale
  - 16-bit data output [72]
- Analog Devices ADXL377 Accelerometer
  - ± 200g linear acceleration full scale [73]
- NXP MPL3115 Pressure Sensor
  - Pressure Scale: 20 kPa to 110 kPa absolute (20-bit)
  - Calibrated temperature output: −40 °C to 85 °C (12-bit) [74]
- Microhard n920 Radio
  - Frequency: 902 – 928 MHz
  - Adjustable Trasmit Power: 100mW to 1W
  - Wide Operating Temp (-40C to +85C) [75]
- ON Semiconductor CAT24C32 Electrically Erasable Programmable Read-Only Memory (EEPROM) (not pictured)
  - 32Kb non-volatile memory
  - 100 Year Data Retention [76]

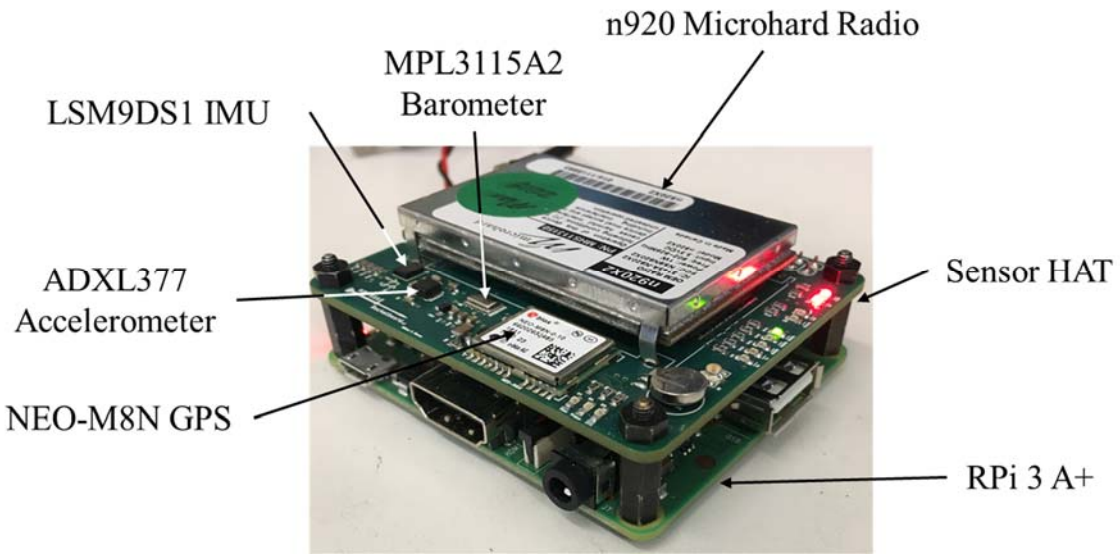


Figure 53. Custom Sensor HAT and RPi 3 A+

The communication protocol for the sensors and components on the custom sensor board HAT is displayed in Figure 54. It can be seen from the image that the majority of the sensors communicate with the RPi via the inter-integrated circuit (I^2C) main bus. The n920 radio is connected to the RPi's one general purpose input and output (GPIO) header serial port and the GPS module communicates using the serial peripheral interface (SPI). Lastly, the EEPROM is communicates with the RPi via an I^2C line specifically reserved for EEPROM operations. I^2C was chosen as the main communication protocol for its simple wiring configuration and ease of implementation with the RPi. The GPS module was connected to the SPI interface of the RPi to separate the relatively large GPS data demands from the other sensors. This allowed for rapid communication on the I^2C line with the high-fidelity sensors.

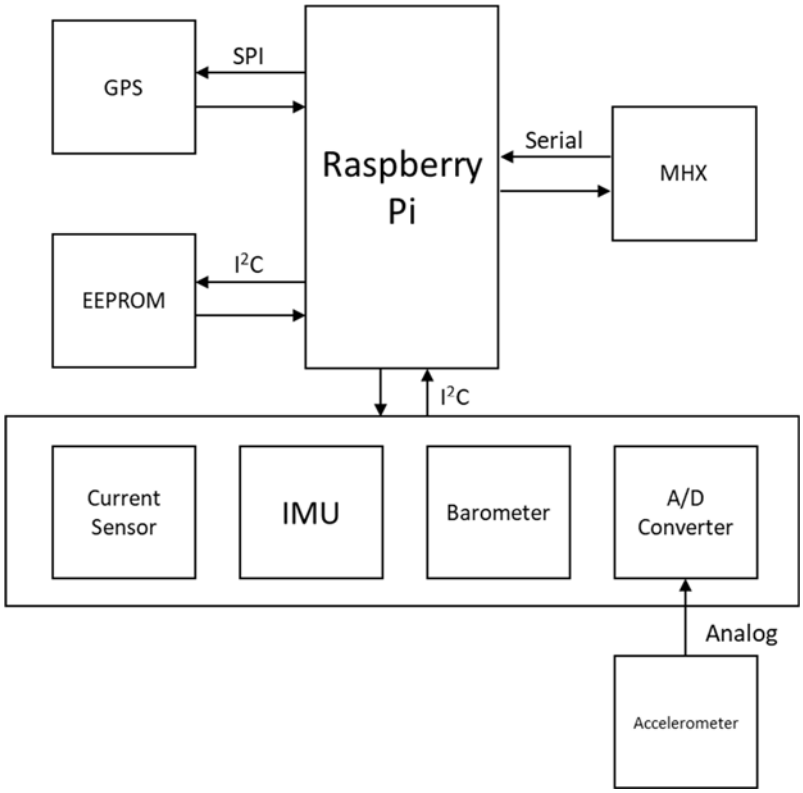


Figure 54. Sensor Communication Protocol

Figure 55 displays a simplified flow chart for the custom sensor board software logic. It can be seen from the chart that the software is kept relatively simple for the purposes of this thesis. Future work on the sensor board could incorporate logic for actionable events as determined by the processed flight data to make the sensor board the primary flight computer and data collector.

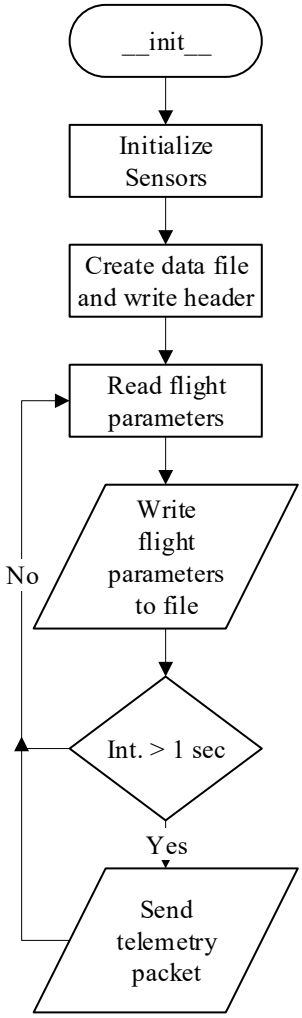


Figure 55. Simplified Software Flow Diagram for Custom Sensor Board

E. COST

The total non-recurring cost to build and fly all components of the NPS SSAG high-power rocket two-stage configuration is approximately $25,000. Assuming no damage to the rocket, subsequent recurring costs for each additional launch total approximately $8,000. The total non-recurring cost to build and fly all components of the NPS SSAG high-power rocket single-stage sustainer section is approximately $6,000. Assuming no damage to the rocket, subsequent recurring cost for each additional launch is approximately $1,900. These figures assume the use of the largest, most expensive

Q15781 and O3400 motors. Cost can be reduced by using smaller, less expensive motors, depending on the mission requirements for altitude.

F. CHAPTER CONCLUSIONS

The initial design of the NPS SSAG high-power rocket is one that enables high mission versatility at a moderate price point for a university program. Its two-stage configuration permits a wide range of altitudes and flight profiles that are tailorable for mission-specific goals. In addition, its incorporation of the CubeSat form factor facilitates payload standardization and subsequently increases responsiveness by decreasing mission preparation time. Lastly, the minimum-diameter design increases the rocket's potential altitude and mission profile variability by minimizing the rocket's weight and cross-sectional area.

V. SUSTAINER TEST FLIGHTS (FLIGHTS 5 AND 6)

On February 23, 2019, a proof-of-concept flight of the rocket's sustainer was conducted to validate its design before proceeding to construction of the rocket's booster. This 5th launch was conducted at the FAR Rocket Range and the motor for flight was the Cesaroni O3400. Onboard the rocket was a narrowband communications relay 2U CubeSat demonstration payload developed by Major John Pross, USMC [77].

The planned profile for the flight was the sustainer-only configuration depicted in Figure 28. In this configuration, the O3400 rocket motor would propel the rocket to an approximate altitude of 35,000 ft. (11 km) AGL. At apogee, the CO_2 ejection system would deploy the CubeSat payload and rocket's drogue parachute. Approximately six minutes after apogee, the rocket would descend to 1,500 ft. (460 m) where the black powder ejection system would be initiated, and the sustainer's main parachute deployed.

The launcher used for this launch was the "Newman Launch-Rack" created by John Newman of the FAR Rocket Range. The rack is 20 ft. (6.1 m) in length and capable of accommodating rockets of diameter between four and twelve inches (10 – 31 cm). The rack was chosen for its adequate length (Chapter III Section C 2.1 lesson learned) and four-rail design which does not require the use of rail buttons, thus allowing for optimization of the rocket's aerodynamic profile. Figure 56 depicts the Newman Launch-Rack in its vertical configuration. Of note in the image are the four vertical aluminum bars that are used to guide the rocket during its initial stages of launch.

Figure 56. Newman Launch Rack at the FAR Rocket Range. Source: [53].

Because the nose cone was larger than the airframe of the rocket, a 3D-printed guide was developed for the aft end of the 4 in. (10 cm) sustainer. Depicted in Figure 57, two sections of the rear sustainer guide are designed to fit around the airframe and connect via the locking tabs and connection grooves. The guide is held in place by the compression of the launch rack guide rails during launch and designed to fall off upon leaving the rack. The guide was placed in front of the fins to mitigate the risk of airframe slipping through the guide. Figure 58 depicts the launch support team loading the rocket into the launch rack in preparation for flight. Of note in the image is the placement of the rear sustainer guide in relation to the fins. It can be seen in the image that the guide is in front of the fins and the launch rack's aluminum guide rails constrain both the guide and nose cone of the rocket.

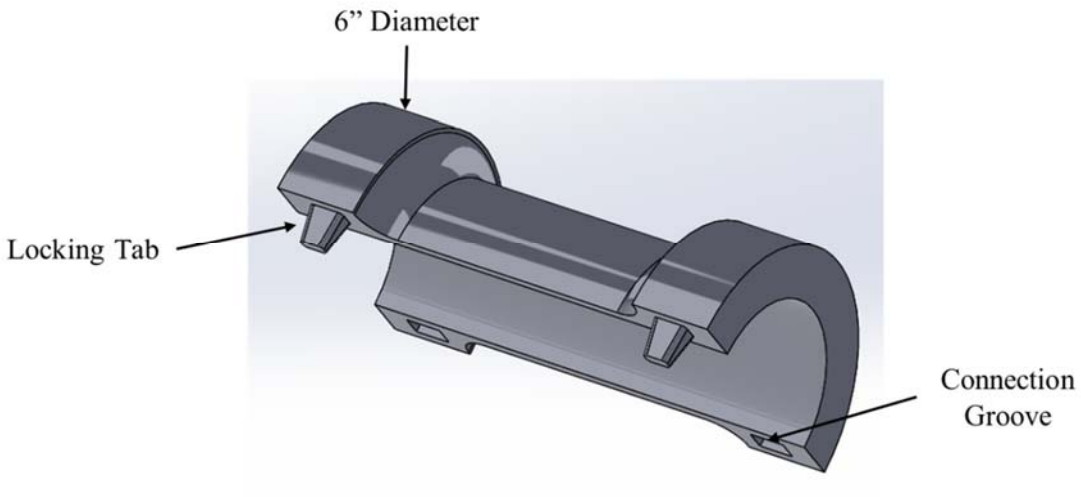


Figure 57. CAD Rendering of Rear Sustainer Guide (Rear Fin Guide.SLDPRT)

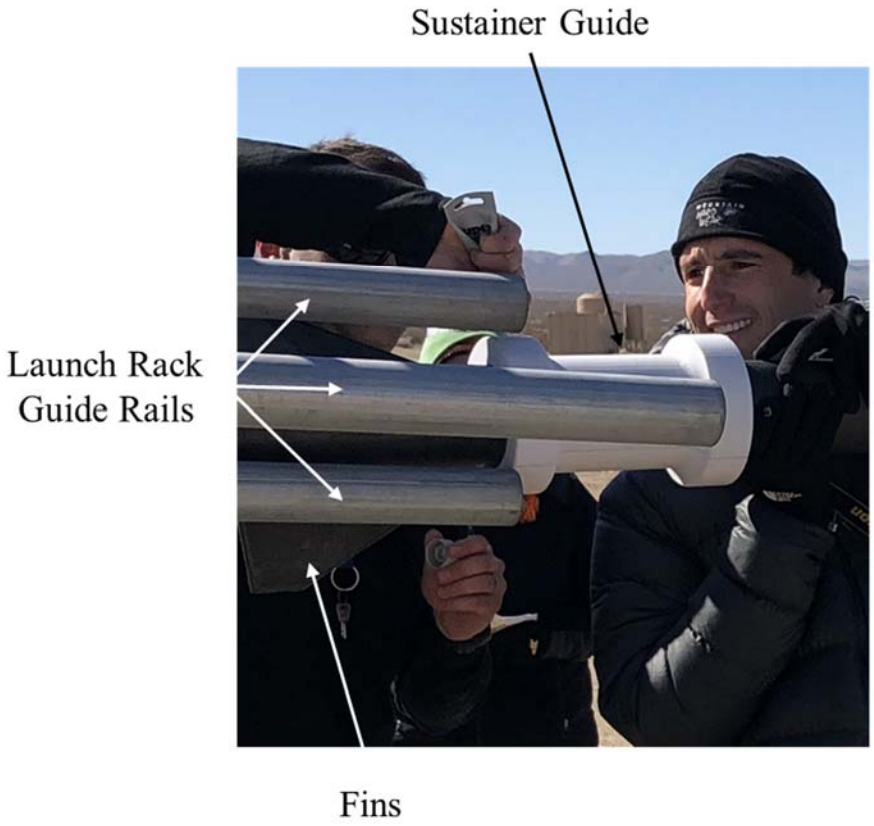


Figure 58. Loading of Sustainer Test Flight Launch

A. FLIGHT 5

The flight of the NPS SSAG high-power rocket sustainer test lasted approximately three seconds before the rocket experienced a catastrophic failure during the transonic region of flight. The anomaly was initially detected at approximately 3,000 ft. (0.91 km) AGL when the rocket experienced an approximate 90° deviation from its initial vertical trajectory. Figure 59 is an annotated video screen capture of the rocket's launch that displays the initial flight path deviation. In the image, the rocket's smoke trail highlights the sharp turn in the rocket's flight path from its initial launch trajectory.

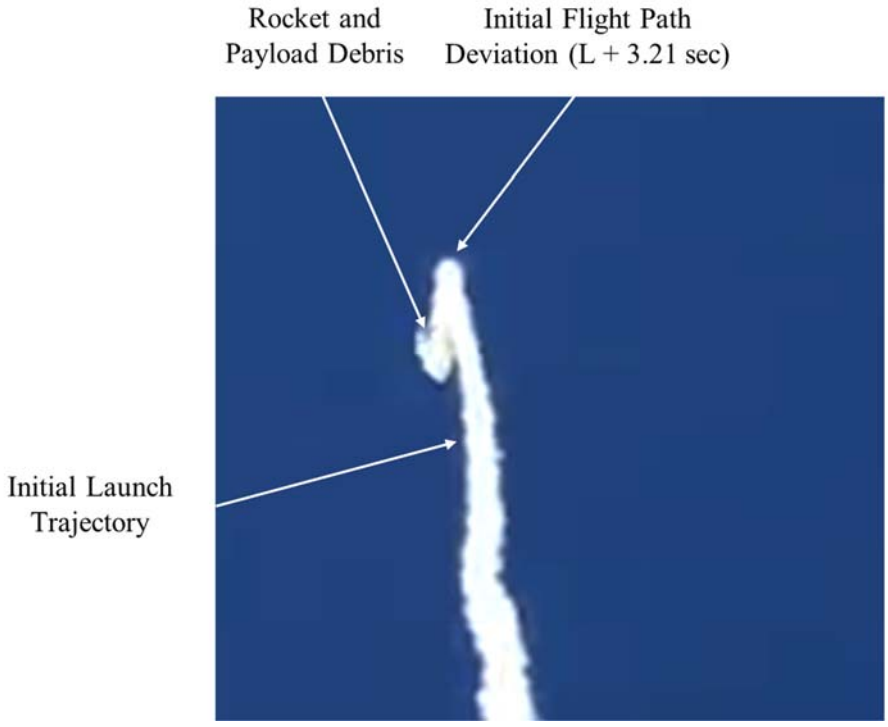


Figure 59. Sustainer Test Flight Initial Flight Path Deviation

Following the initial deviation from its launch trajectory, the rocket continued to make a series of approximately three sinusoidal course deviations from the previous stable flight conditions. Debris from both the rocket and payload were observed falling from the main portion of the airframe throughout the remainder of the motor burn period. Figure 60 is an additional annotated video screen capture of the rocket's launch later in

its flight profile. In the image, the multiple sinusoidal course deviations and rocket and payload debris can be observed. At the motor burnout portion of flight, all sections of the rocket and associated debris began falling back to the ground. Once all debris was observed to have impacted the ground, recovery operations were initiated.

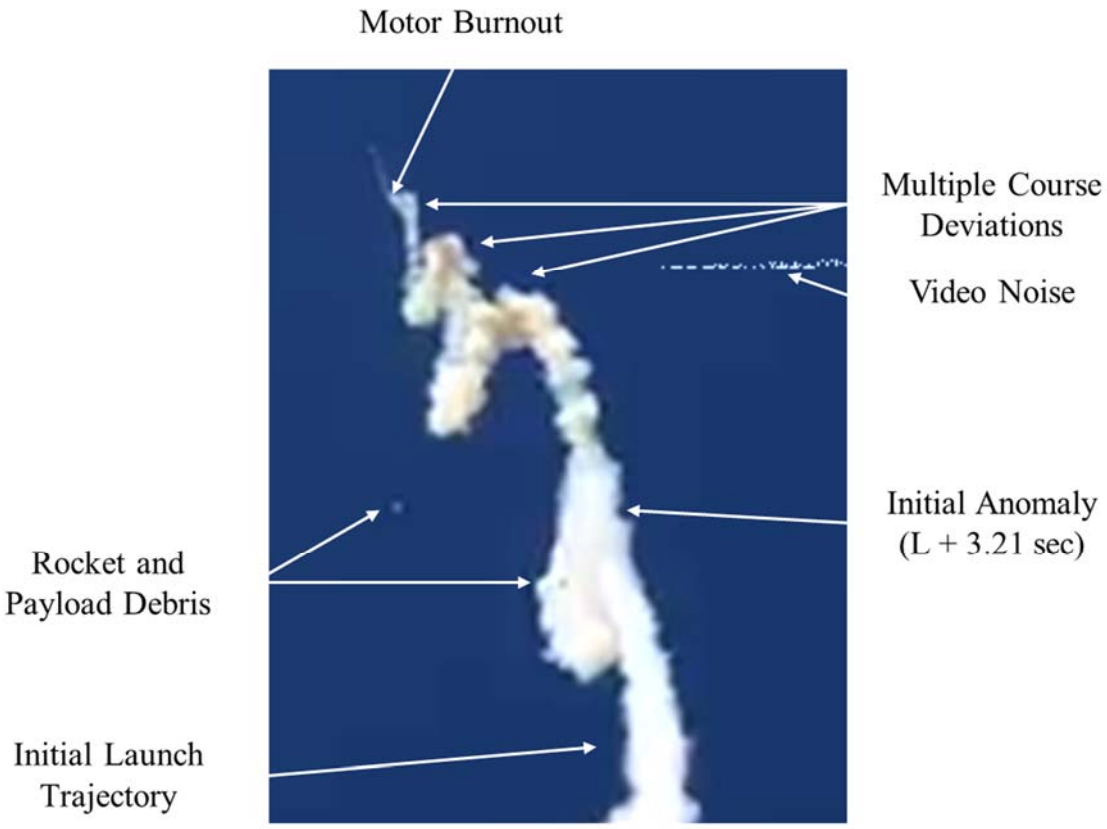


Figure 60. Sustainer Test Flight - Flight Path

Despite the deviation from nominal flight, the nose cone survived and fell to the ground under its parachute. All electronics housed in the nose cone were still functional and the associated flight data was recovered. Figure 61 displays the GPS data from the BigRedBee transmitter in a KML rendering. The figure shows the off-nominal flight trajectory and the parachute descent of the nose cone section.

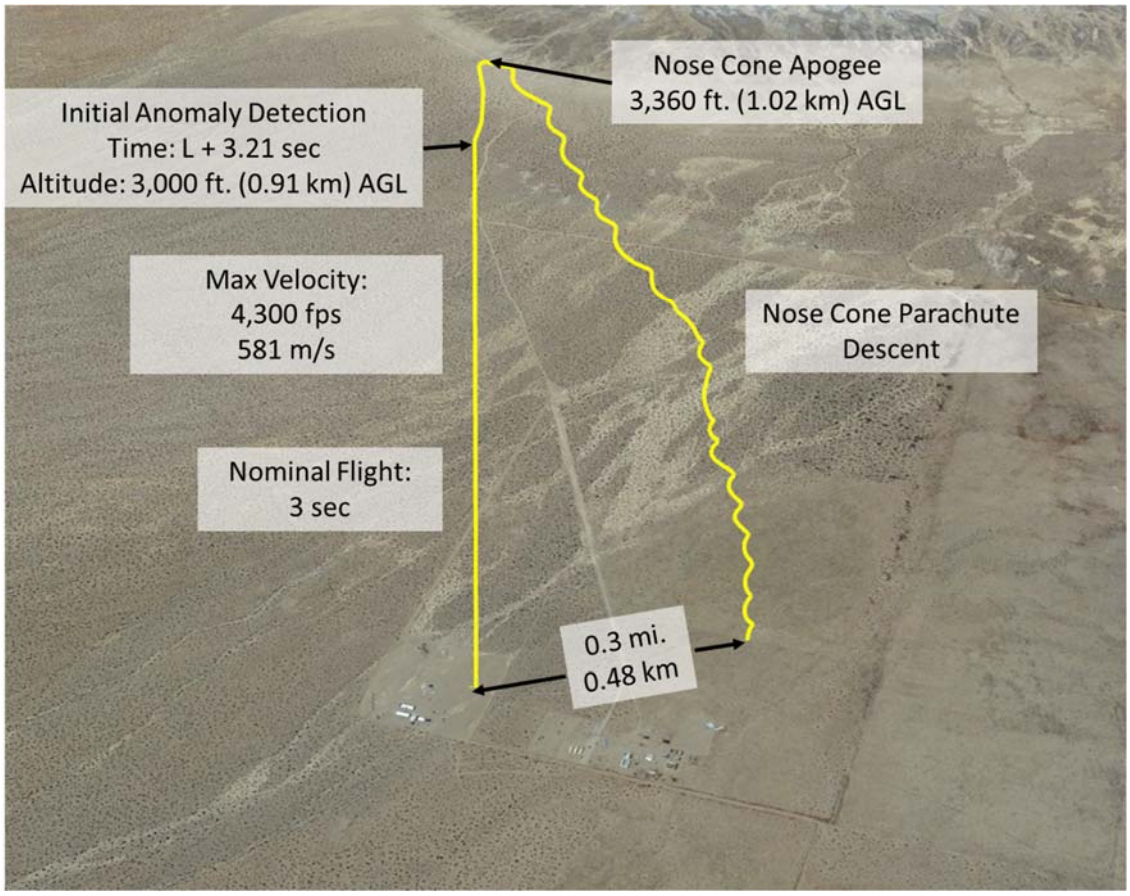


Figure 61. BigRedBee GPS Data KML File Rendering

Unfortunately, the deployment of the payload at approximately Mach 1.5 resulted in unplanned and rapid disassembly of the CubeSat, preventing the collection of any experimental payload data. Some of the pieces of the payload were recovered during a search of the debris field. Figure 62 shows a before and after photo of the payload telemetry, tracking, and control (TT&C) communications ground plane, a piece of 1/32 in. (0.79 mm) aluminum plate attached to the polycarbonate CubeSat frame.

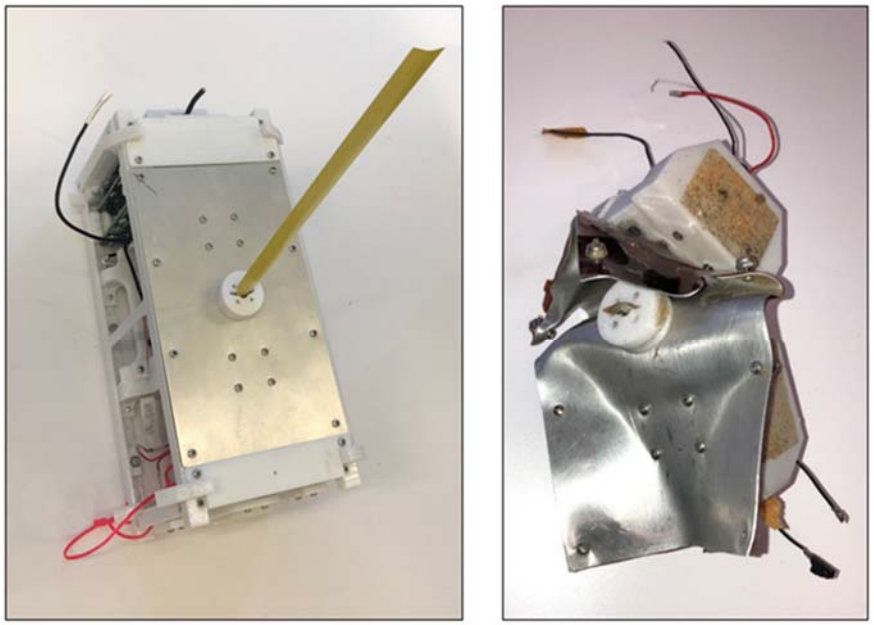


Figure 62. Payload TT&C Ground Plane Before and After. Adapted from [77].

B. POST-FLIGHT INVESTIGATION ANALYSIS

Prior to conducting the post-flight anomaly investigation analysis, the project support team recovered the larger pieces of the rocket and payload debris. Figure 63 annotates where some of the larger pieces of the rocket were found and where the majority of the debris field was located in the red marked area. In addition, the approximate flight path of the rocket is denoted by the yellow arrow in the figure.

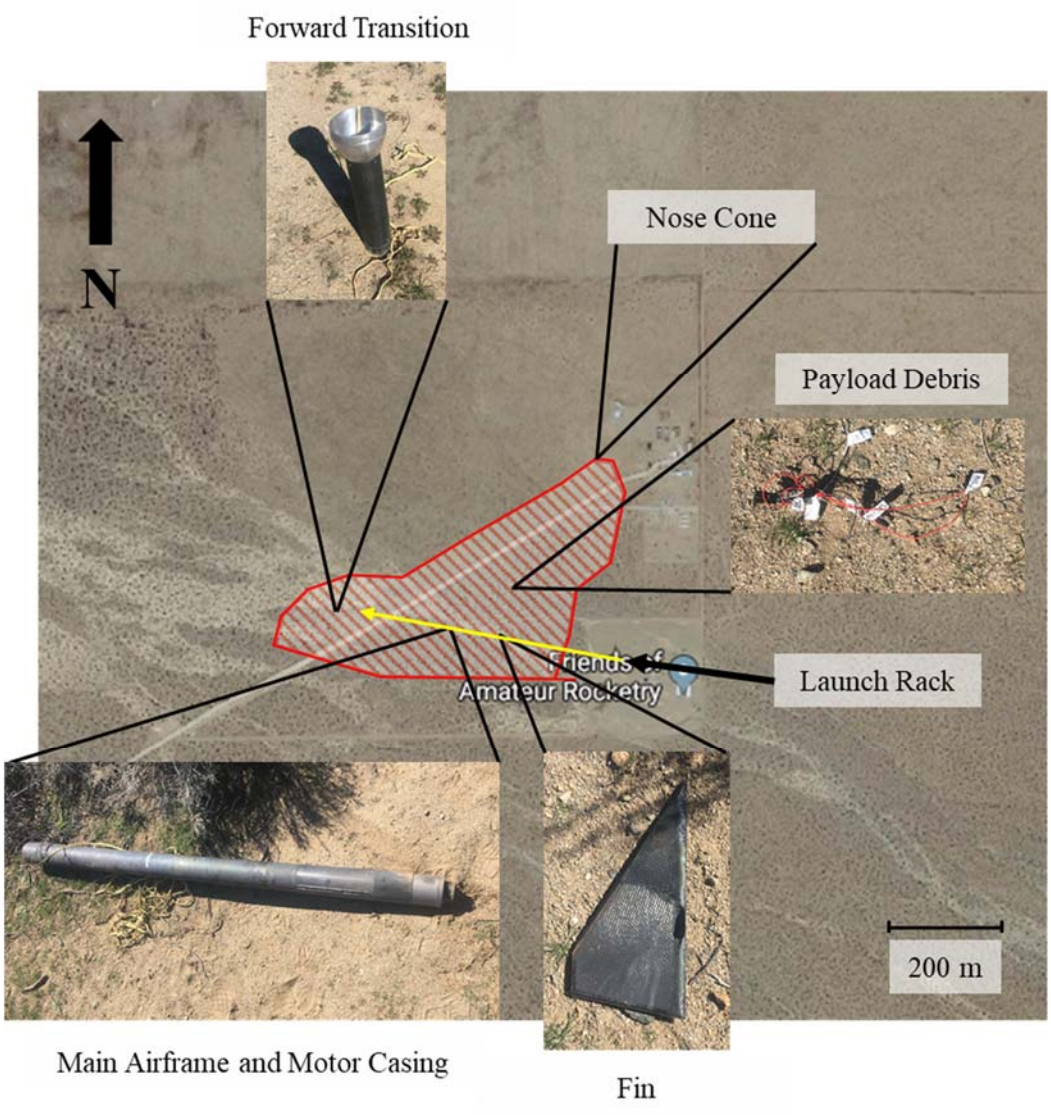


Figure 63. Rocket and Payload Debris Field

Examination of the image reveals most of the larger, heavier rocket parts such as the motor casing and forward transition impacted in the southern and western regions of the debris field along the rocket's flight path. In contrast, the lighter pieces of debris, and those parts under parachute such as the payload wiring and rocket nose cone, were carried by the wind to the northern and eastern parts of the debris field. The location of the detached fins suggests they may have separated from the rocket early in the flight, causing the initial flight deviation and perhaps precipitating the subsequent detachment of

the nose cone and premature deployment of the payload. The initial assumptions that led to this conclusion and subsequent data analysis that shows this may not have been the case are discussed in the following sections.

1. Initial Assumptions

Prior to the rocket's flight, there was concern from project advisors about the 3D-printed sustainer guide compromising the integrity of the fin-airframe joint. Because the guide was placed in front of the fins, it was suggested that guide had the potential to damage the fins during launch, thus affecting the rocket's stability. This theory was reinforced the day of the launch when upon closer examination of the aluminum guide rails, it was observed that they were not continuous aluminum rods, but rather composed of two joined sections. This setup left a small gap between the two sections of the rail, pictured in Figure 64. When sliding the rocket and guide across this section, the leading edge of the sustainer guide had the potential to get caught in the small gap. To mitigate this issue, the leading edge of the sustainer guide was chamfered to reduce the impact angle and decrease the potential for the guide to get caught.

Figure 64. Launch Guide Rail Gap

2. Supporting Evidence

Initial evidence supported the theory that the sustainer guide damaged the fins and initiated the catastrophic flight path deviation. The first piece of evidence that supported this conclusion was the location of where the relatively light fins were found in the debris field. As depicted in Figure 63, all of the fins were discovered near the initial trajectory path and in the early stages of the rocket launch. This location in the debris field indicates the fins detached from the main airframe of the rocket early in the deconstruction process and did not achieve the same altitude as some of other components. As such, their drift in the wind was limited and they remained near the initial stages of the flight path. In addition, upon examinations of the fins, it was discovered that there was evidence of impact damage on the leading edge of one of the fins at the point where the sustainer guide rested. Figure 65 shows this impact damage as it was discovered on the recovered fin.

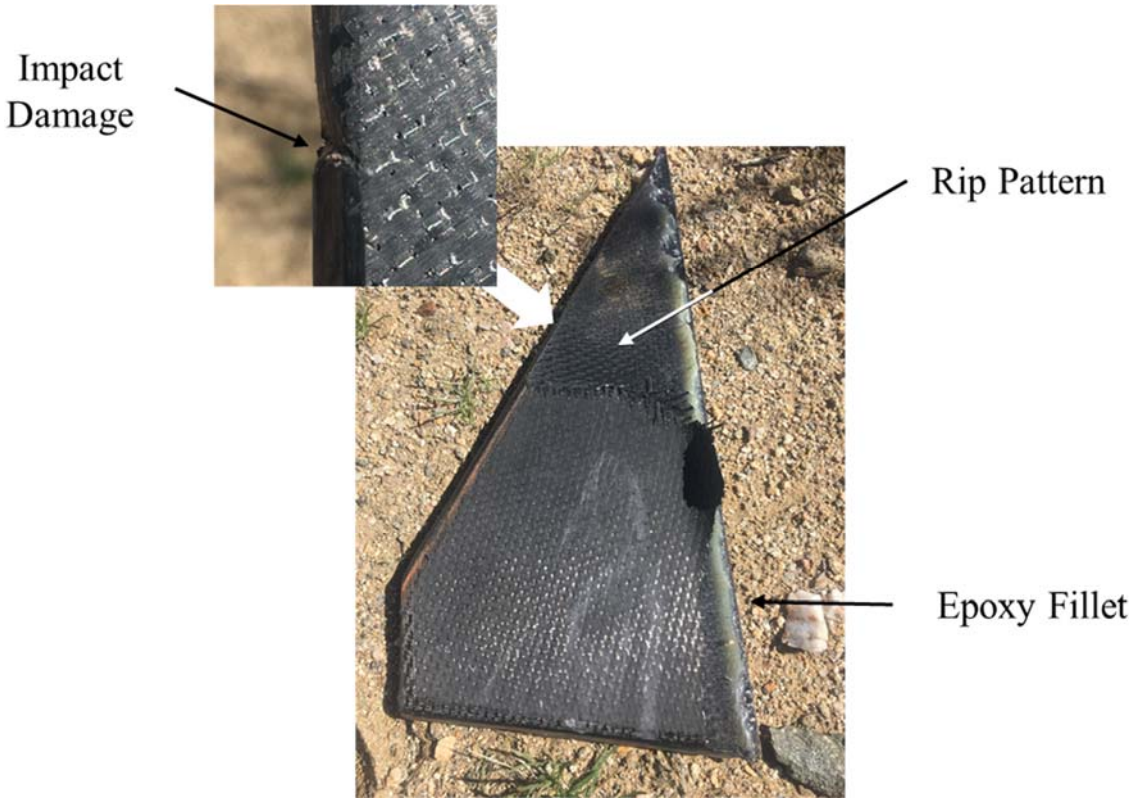


Figure 65. Sustainer Fin Impact Damage

However, upon processing of the custom sensor board data from the nose cone, it was discovered that there may have been an alternative reason for the catastrophic failure. Sensors onboard showed the anomaly more precisely occurred between 3.21 and 3.22 seconds after launch. The anomaly detection is highlighted in Figure 66 and Figure 67 which display the processed accelerometer and gyroscopic data from the custom sensor board, respectively. On both of the plots, the different phases of the nose cone's flight are displayed.

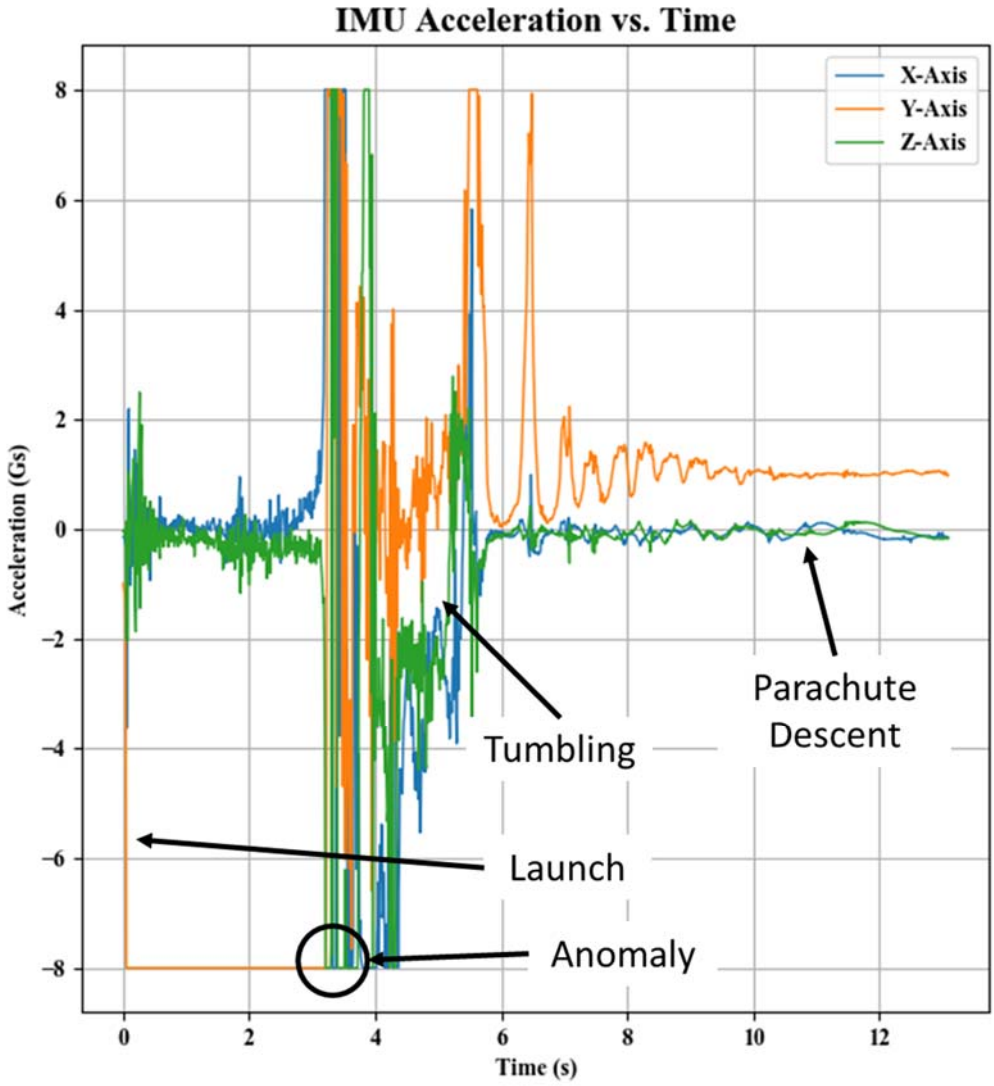


Figure 66. Sustainer Test Flight Accelerometer Data

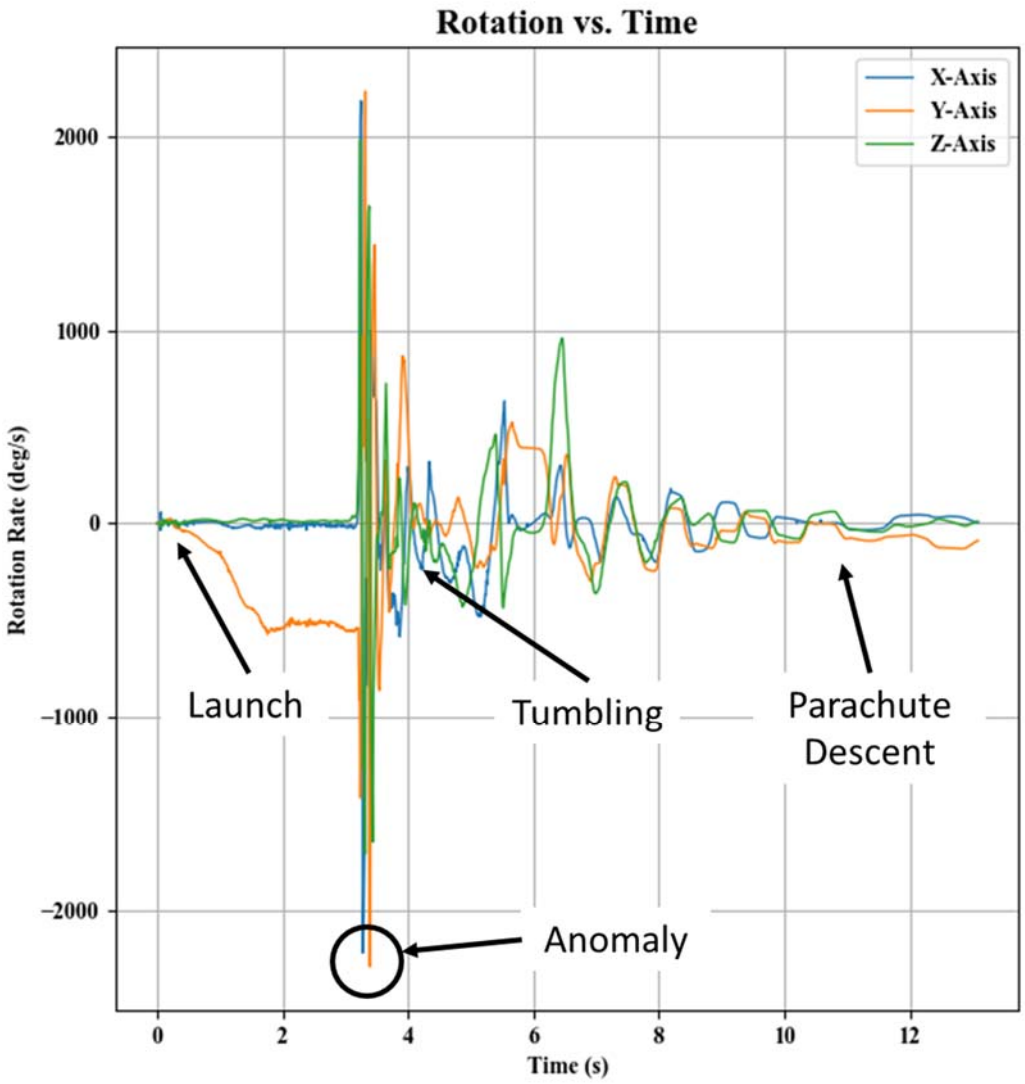


Figure 67. Sustainer Test Flight Gyroscope Data

Cross correlation of the time of anomaly with flight footage reveals the nose cone falling off the airframe of the rocket prior to any significant deviation in the flight path angle. Figure 68 displays a screen capture of the flight video at the time the anomaly was indicated on the nose cone flight sensors. In the image, what is most likely the nose cone can be seen under the main airframe of the rocket prior to a signficant flight path deviation.

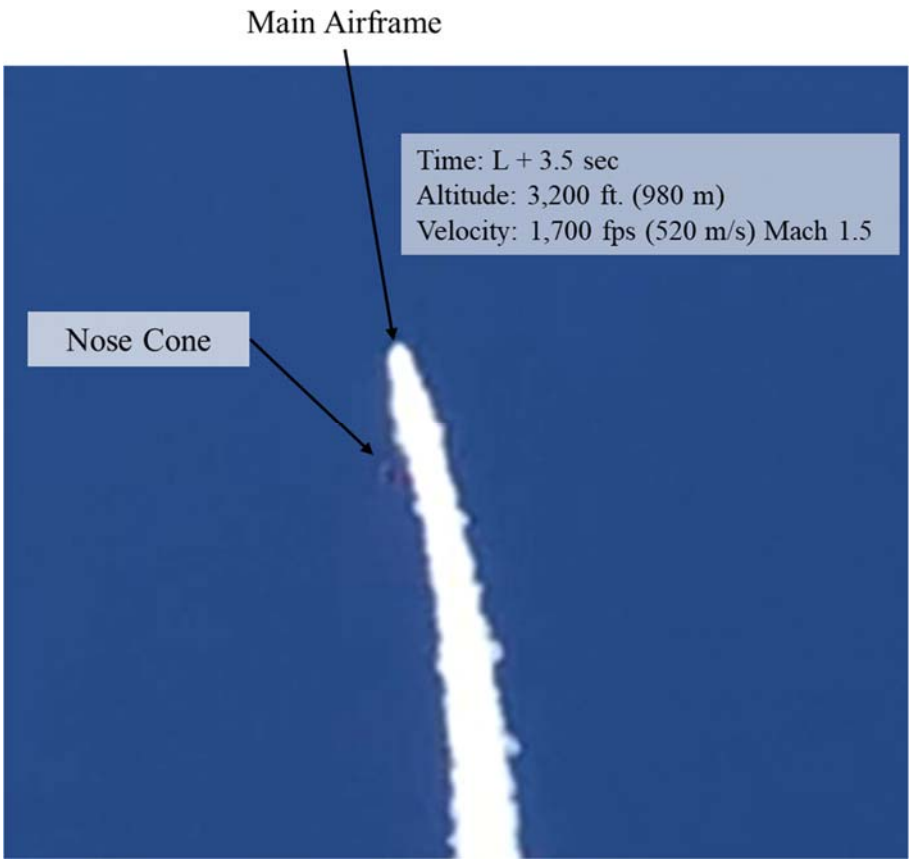


Figure 68. Post-flight Video Analysis Anomaly Detection

Figure 69 shows the anomaly in a frame-by-frame video view as it occurred. In frame one, the rocket can be seen following its initial vertical trajectory. In frame two, the initial course deviation is evident (the apparent plume of smoke is actually the rocket turning, the angle of the video is such that the developing smoke trail is masked by the lingering initial trajectory smoke trail). In frame three, what is most likely the nose cone can first be seen falling from the rocket. In frames 4, 5, and 6 the nose cone continues to fall from the rocket and payload debris is observed.

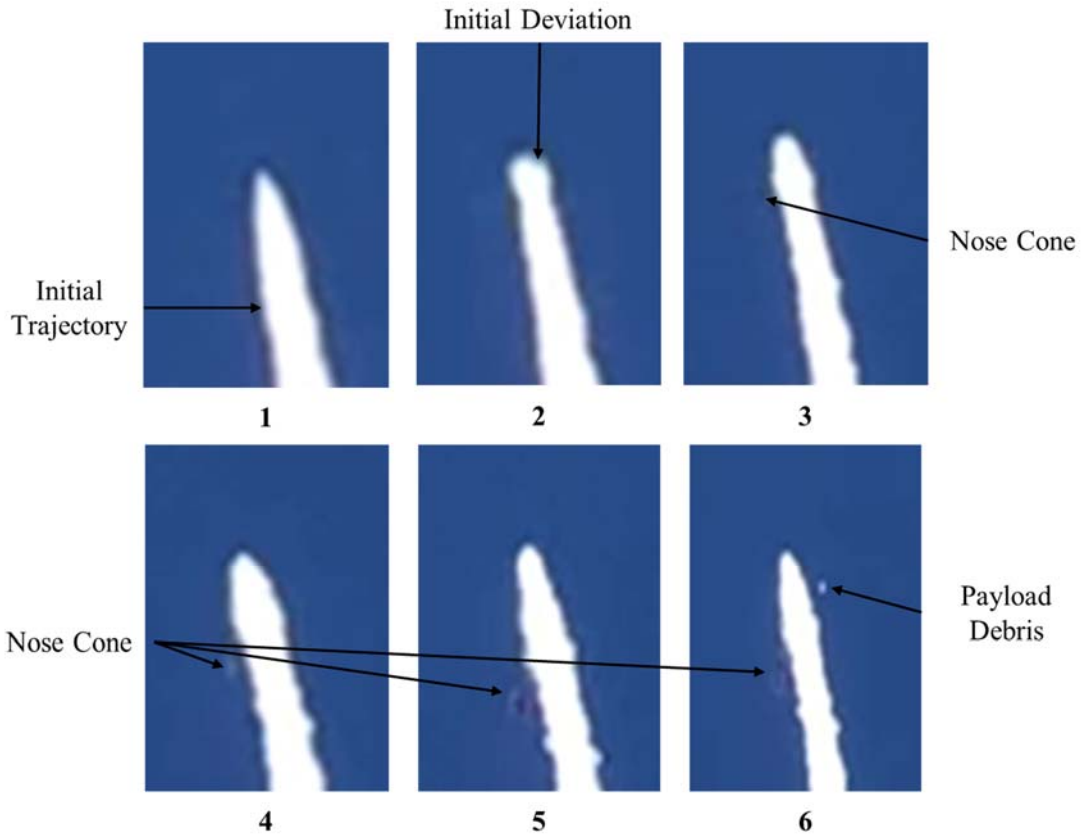


Figure 69. Frame-by-Frame View of Test Flight Anomaly

The correlation of the nose cone sensor data and flight video indicate the nose cone fell off prior to the rocket becoming unstable. Though the fins may have been damaged by the 3D-printed sustainer guide, the data suggests it was the loss of the nose cone that caused the initial flight path deviation. Therefore, it is likely that the subsequent course deviations observed in Figure 60 were the result of the remaining fins altering the rocket's orientation in the aerodynamic stream and systematically being torn from the airframe by the associated forces. The following is a summary of the potential reasons for the nose cone's detachment:

1. Insufficient overlap between the nose cone and transition shoulder, resulting in aerodynamically-driven torques on the nose cone that led to failure of the attachment to the transition shoulder.

2. Insufficient venting of the payload area. This could have resulted in a rapid buildup of air pressure in the nose cone which may have separated the nose cone and precipitated the premature payload deployment.

3. The CubeSat payload, though relatively constrained in the nose cone, was not locked into position. Perhaps the launch vibration resulted in resonance vibration of the payload which forced the nose cone out of alignment with the airstream, resulting in aerodynamically-driven torques on the nose cone that led to failure of the attachment to the transition shoulder.

3. Anomaly Investigation Conclusions

From the post-flight anomaly investigation, it appears evident that it was the nose cone's detachment from the forward transition that was the catastrophic failure. The initial assumption as to the cause of the nose cone's detachment was that there was insufficient venting of the nose cone region (Sec. B 2.2) and the resulting pressure differential between the internal compartment of the nose cone and the ambient air at the altitude at which the nose cone detached was enough to generate a bulkhead force greater than the shear pin strength. However, post-flight analysis reveals that this was likely not the case.

By rearranging Equation 3, the internal bulkhead force generated on the nose cone can be solved for by multiplying the nose cone's bulkhead surface area by the internal and exterior pressure differential. The pressure differential between the internal compartment of the nose cone and the ambient air at the altitude at which the nose cone detached was approximately 1.5 psi (10,300 Pa). Additionally, the nose cone bulkhead surface area was 28 in^2 (0.018 m^2). Multiplying these two numbers results in a bulkhead force of 42 lbf (190 N). According to [65], the average shear strength of the 2–56 nylon screws that retained the nose cone to the aluminum transition is 39 lbf (170 N). Given that two 2–56 screws were used in total, the total shear strength of the nylon screws was 78 lbf (340 N). This calculations shows that the bulkhead force generated by the pressure differential between the inside of the rocket and outside atmosphere was not significant

enough by itself (by a margin of 44%) to overcome the shear strength of the 2 x 2–56 shear pins retaining the nose cone to the transition. Therefore, insufficient venting of the nose cone was not the reason for the nose cone's detachment.

Instead, it is most likely that the nose cone separated from the forward transition due to aerodynamically-driven torques induced by lateral dynamic pressure caused by misalignment of the nose cone to the direction of travel of the rocket. As there was insufficient overlap between the nose cone and forward transition, these aerodynamically-driven torques were sufficient enough to overcome the strength of the nose cone-forward transition joint. The result was the detachment of the nose cone and subsequent premature deployment of the CubeSat payload.

4. Flight 5 Lessons Learned

Several lessons were learned from the flight and post-anomaly analysis of flight 5:

1. There should be significant overlap between each section of the rocket. It is recommended that at least ½ the diameter of airframe be overlapped with the adjoining segment. Increasing section overlap enhances the structural integrity of the rocket and mitigates the risk of rocket segments coming apart due to the forces experienced during flight.

2. Payloads should be secured tightly in the rocket airframe. Though it cannot be confirmed that the resonant vibration of the payload forced the nose cone out of alignment, resulting in the failure of the attachment to the transition shoulder, it is possible. To mitigate this issue for future flights, payloads should be better secured to the rocket airframe.

3. Vent holes need to be checked for flow obstruction prior to flight. Ensuring that the vent holes are not covered by recovery hardware or parachutes during the final assembly of the rocket will ensure the vent holes function as designed.

4. Video of the launch needs to be taken in high definition and from multiple angles. This will aid in the post-flight data analysis for all flights.

C. SUSTAINER RE-DESIGN

As a result of the catastrophic failure that occurred during the sustainer test flight, and given the short amount of time to plan, build, test, and conduct another flight, the NPS SSAG high-power rocket sustainer section was redesigned. To mitigate the complexity and decrease the overall program cost, the ability to accommodate the CubeSat form factor was temporarily removed from the design. In addition, to simplify the construction process, the airframe material of the stage was changed to fiberglass, more readily available and easier to work with than carbon fiber, and a commercially-available aluminum fin can was incorporated. Figure 70 depicts a CAD rendering of the sustainer redesign. Within the image, the significant changes from the previous design are annotated.

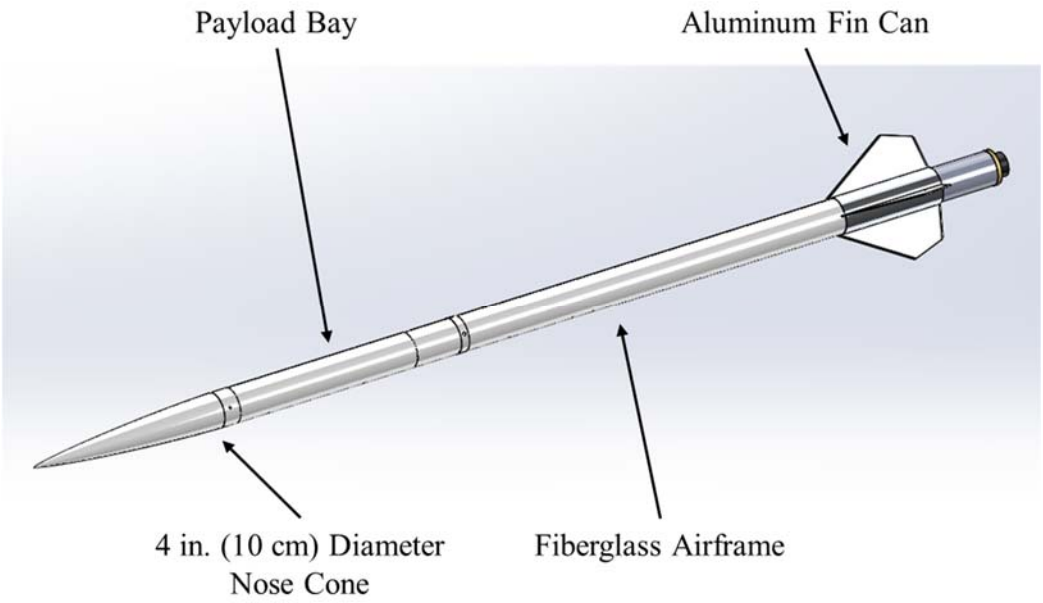


Figure 70. CAD Rendering of Sustainer Redesign

Figure 71 is a prelaunch photo of the rocket with various launch personnel. In the middle of the image is the author, to his left is the primary thesis advisor (Dr. Jim Newman), and to his right is SSAG machinist (Levi Owen).

Figure 71. Prelaunch Photo of Rocket, Author (middle), Thesis Advisor (left), and SSAG Machinist (right)

1. Rocket Characteristics

Figure 72 is a sectional CAD rendering of the sustainer redesign. The figure highlights the major components of the new design and shows their location within the rocket. Of note within the image is the change to the nose cone profile and the addition of the aluminum fin can. All other components within the rocket remain unchanged from the previous design except the change to an all fiberglass airframe.

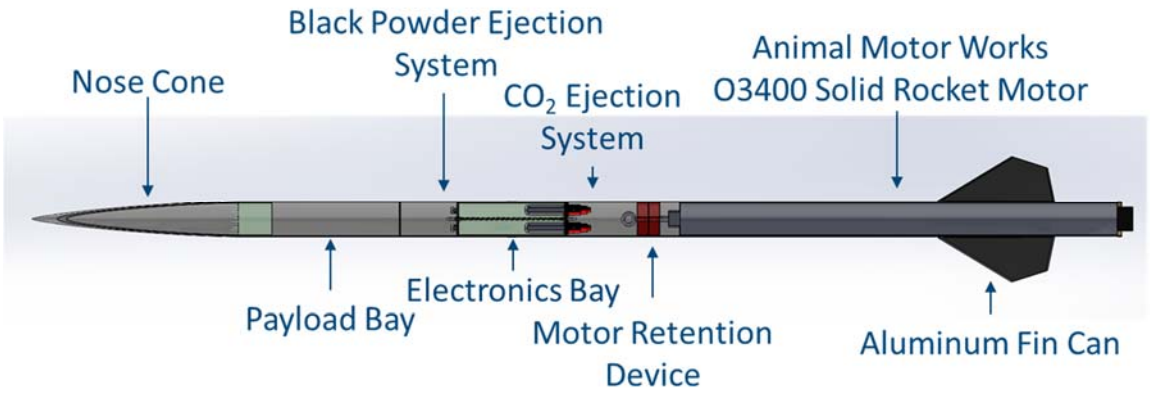


Figure 72. Sustainer Redesign Components

Given the new form factor, the payload size restraints are driven by the dimensions of the payload bay which has a diameter of 3.9” (9.9 cm) and is of variable length. The design of the rocket is such that lengthening of the payload bay to accommodate various sized payloads only marginally shifts the CP. Therefore, it is possible to increase or decrease the size the payload bay depending on the individual mission requirements. This concept is shown in Figure 73 which displays the results of the stability analysis for the sustainer redesign with a 20” (51 cm) and 50” (127 cm) payload bay. Examination of the image shows that the resulting shift in CP from the lengthening of the payload bay is marginal and that the rocket is still stable.

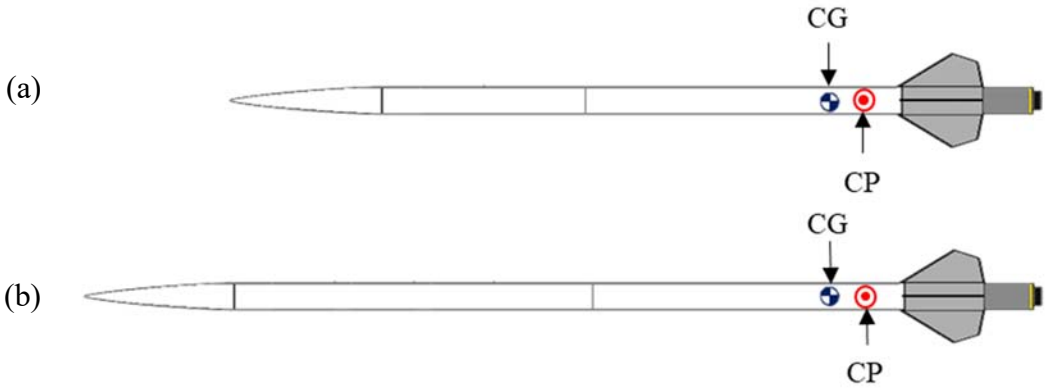


Figure 73. Sustainer Redesign Stability Analysis with (a) 20” (51 cm) Payload Bay and (b) 50” (127 m) Payload Bay

2. Flight Profile

Figure 74 displays an annotated version of the updated flight profile for the sustainer-only flight configuration. It can be seen from the image the projected maximum altitude is approximately the same as the initial design even with the heavier fiberglass airframe. In addition, it is no longer necessary to eject the payload for parachute deployment. Therefore, the payload will remain within the payload bay of the rocket for the duration of the mission. Though the dual-deploy configuration is illustrated in the figure, the main parachute could be deployed at apogee for those payloads desiring additional dwell time.

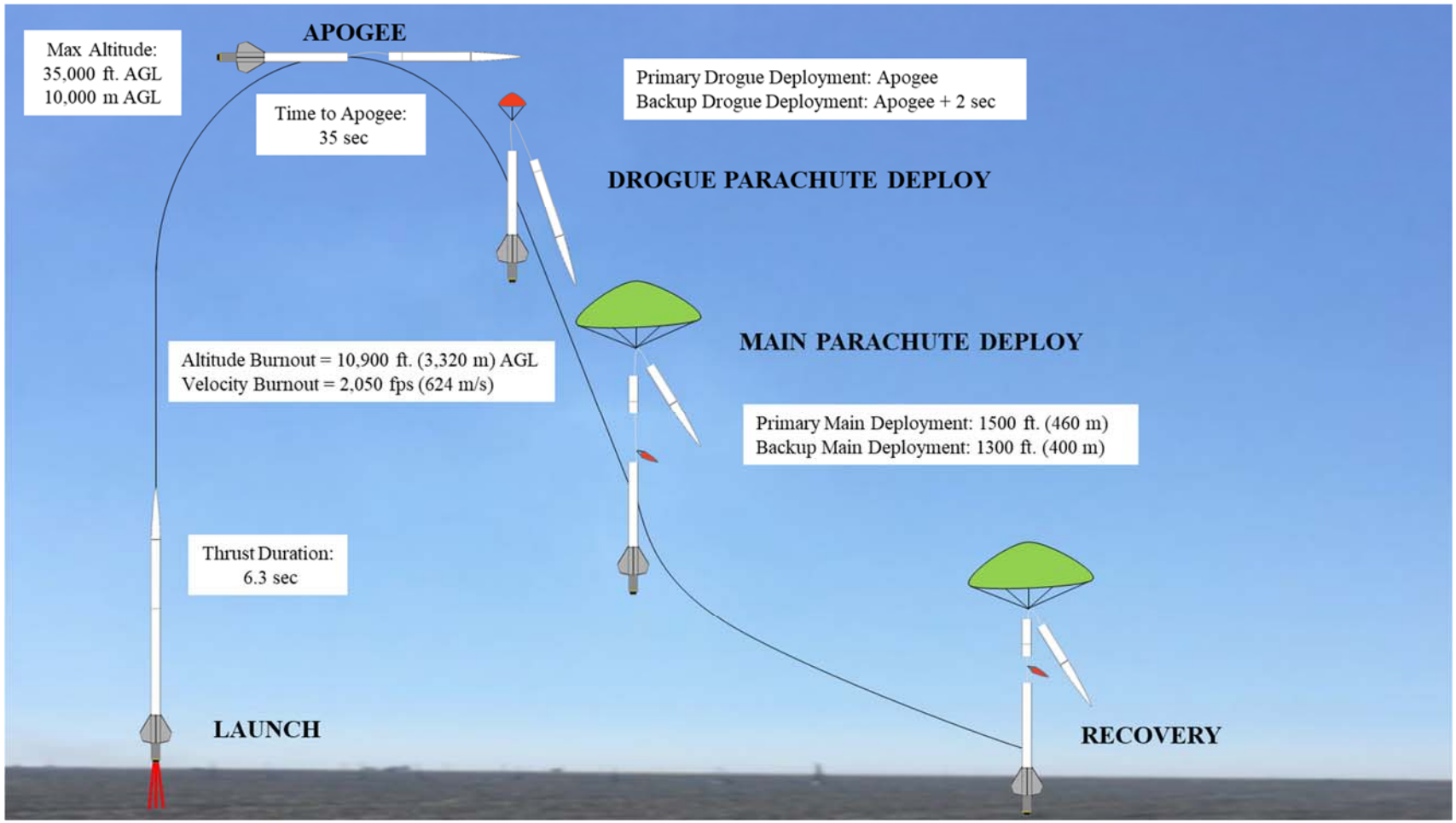


Figure 74. Planned Flight Profile of Sustainer Redesign Launch

3. Cost and Complexity

By forgoing the incorporation of the CubeSat form factor, using fiberglass as the airframe material, and using a commercially available aluminum fin can, the cost and complexity of the rocket was greatly reduced. Instead of $6,000, the total non-recurring cost to build and fly all components of the NPS SSAG high-power rocket single-stage configuration rocket is approximately $3,000. In addition, the complexity of rocket construction and assembly was greatly reduced. In contrast to the two months it took to build and assemble the previous design, the new sustainer rocket was constructed in approximately one week.

4. Pre-flight Analysis

Key flight performance parameters for the sustainer rocket re-design were calculated using both MATLAB and RockSim. Table 18 summarizes the results of the analysis. Examination of the calculated values indicated that the maximum projected height of the rocket, as calculated by both Rocksim and the author, varied by 5%.

Table 18. Flight 6 Pre-flight Analysis Summary

Prediction Method	Maximum Acceleration	Maximum Velocity	Maximum Height - AGL	Time to Apogee
RockSim Calculated	727 ft./s^2 222 m/s^2	2,740 ft./s 835 m/s	36,000 ft. 11,000 m	41 s
Author Calculated [App I Sec. B]	538 ft./s^2 164 m/s^2	2,150 ft./s 657 m/s	38,000 ft. 11,800 m	44 s

D. SUSTAINER REDESIGN TEST FLIGHT (FLIGHT 6)

On May 11, 2019 a proof-of-concept flight of the rocket's sustainer redesign was conducted to validate its design. The launch was conducted at the FAR Rocket Range and the motor for flight was the Cesaroni O3400. The planned profile for the flight was the sustainer-only configuration depicted in Figure 74. In this configuration, the O3400 rocket motor would propel the rocket to an approximate altitude of 35,000 ft. (11 km)

AGL. At apogee, the CO_2 ejection system would deploy the rocket's drogue parachute. Approximately six minutes after apogee, the rocket would descend to 1,500 ft. (460 m) where the black powder ejection system would be initiated and the sustainer's main parachute deployed.

1. Flight

Prior to the launch of the rocket, a final check as part of the prelaunch checklist was conducted on the rocket's stability to ensure the CG was in fact in front of the CP. Upon holding the rocket at its balance point, it was discovered that the estimated CG location did not reflect the designed location without the additional weight of the payload and was in fact behind the Barrowman estimated CP location. To move the CG location forward, an estimated 4 lbs. (1.8 kg) of sand and gravel in Ziploc bags were added to the rocket's nose cone and payload section. The addition of the temporary ballast material had the intended effect and shifted the CG approximately 3 in. (7.6 cm) ahead of the CP. Figure 75 displays a confirmation of the rocket's CP and CG locations with the added ballast. In the image, the Barrowman estimated CP location is depicted by the red sticky and the CG location is at the point indicated by the author's finger. Forward on the rocket is to the right of the image and aft is to the left. It can be seen from examination of the image that the CG location is approximately 3 in. (7.6 cm) forward of the Barrowman estimated CP location and approximately 6 in. (15 cm) in front of the RockSim estimated CP, confirming the rocket's stability prior to flight. Although this gives a margin of 0.75 or 1.5, respectively for the Barrowman and RockSim CP estimates, this proved adequate for flight stability.

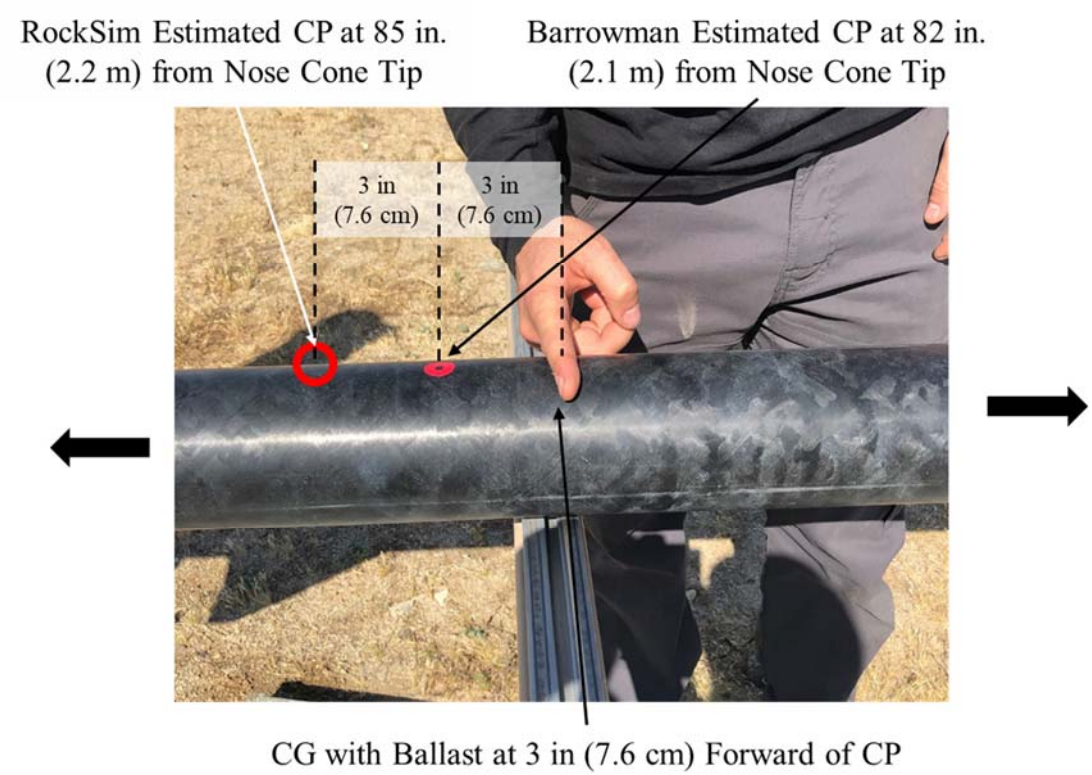


Figure 75. Rocket Stability Confirmation

The flight of the re-designed SSAG high-power rocket sustainer lasted approximately 12 minutes from launch to splashdown and the rocket achieved an approximate altitude of 39,000 ft. (12,000 m) AGL as indicated by the custom sensor board barometer data and TeleMega GPS telemetry data. At apogee, the drogue parachute deployed nominally and the rocket fell at the intended descent rate of approximately 70 fps (21 m/s) and 13 fps (4.0 m/s) under the main parachute. Figure 76 depicts a screen capture from the onboard video of the rocket at apogee. The figure shows the separation of the two rocket sections, confirming the deployment of the drogue parachute.

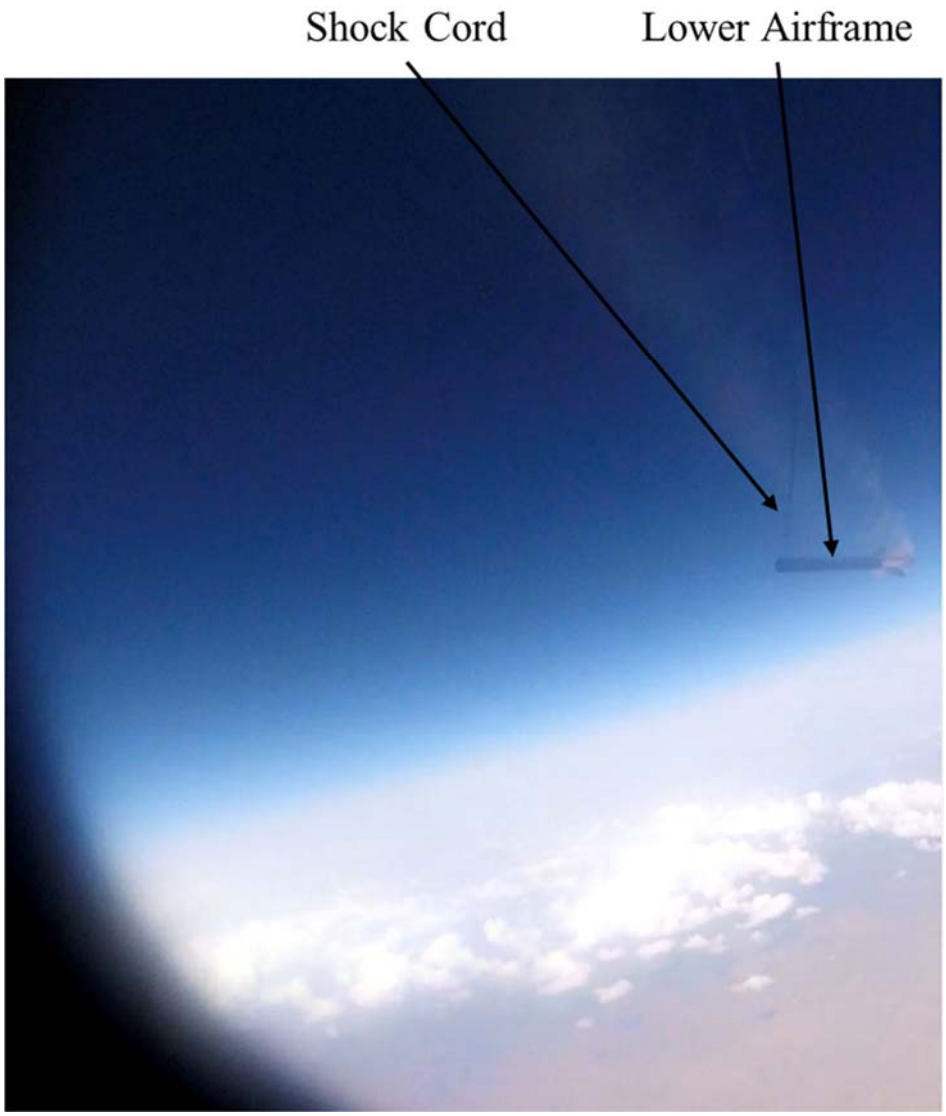


Figure 76. Screen Capture at Apogee of On-board Rocket Video

The rocket then descended to an altitude of 1,500 ft. (460 m) where the main parachute also deployed nominally. Figure 77 is a graph of the altitude profile generated by the custom sensor board barometer data. In the figure, the rocket's descent under drogue parachute and the main parachute deployment events are highlighted. The noticeable change in the rocket's descent rate confirms nominal function of the main parachute. The rocket landed approximate 6.2 miles (10 km) from the launch site as indicated by the BigRedBee GPS telemetry data.

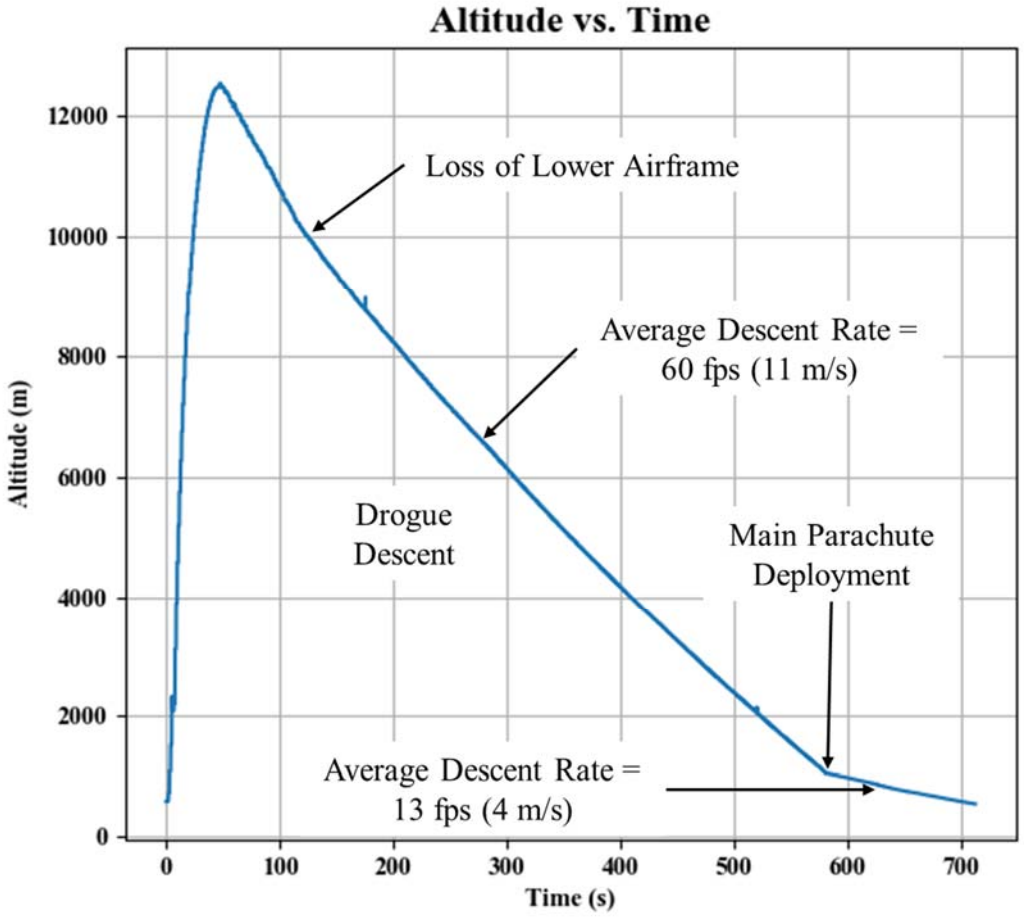


Figure 77. Sustainer Test Flight Barometer Data Altitude vs. Time Graph

Figure 78 is a graph of the rocket's velocity as a function of time generated from the barometer data altitude readings. In the figure, the various stages of flight can be seen as well as the rocket's approximate maximum velocity of 2,790 fps (850 m/s). Figure 79 displays an enlarged portion of Figure 78 during the last three minutes of the rocket's flight.

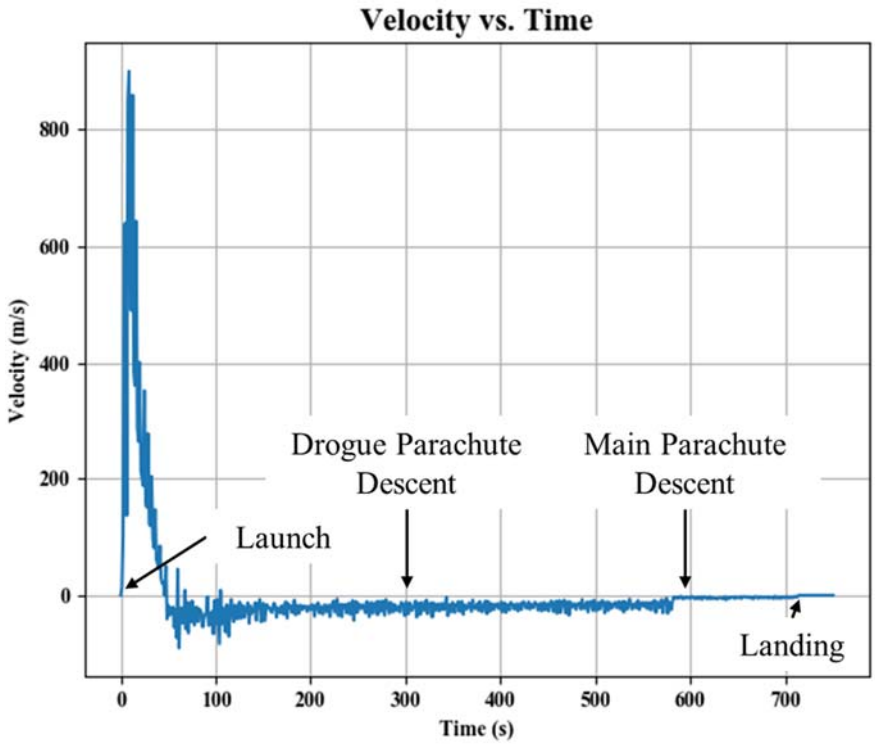


Figure 78. Sustainer Test Flight Barometer Data Velocity vs. Time Graph

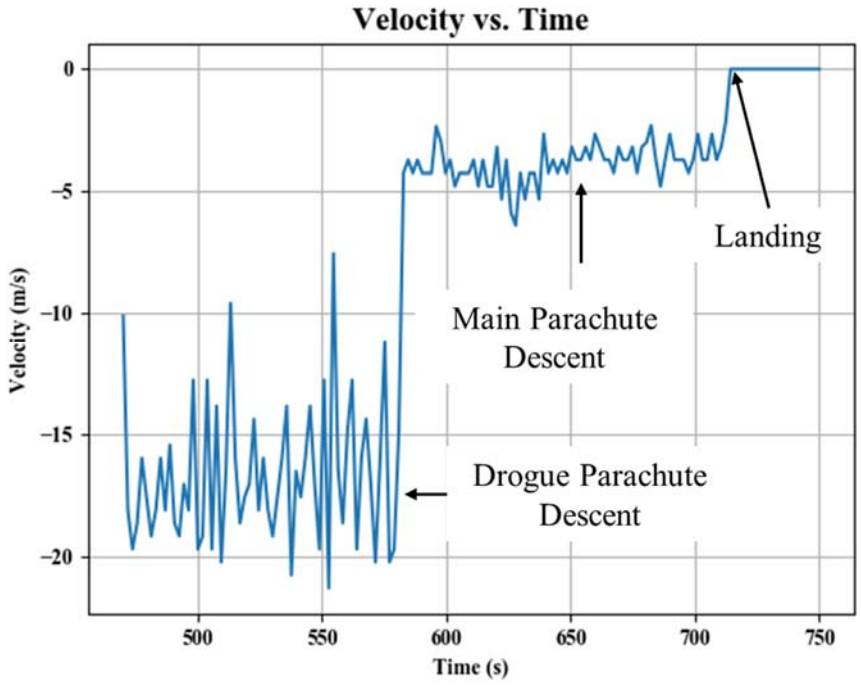


Figure 79. Sustainer Test Flight Barometer Data Velocity (End of Flight) vs. Time Graph

2. Recovery

Due to rain in the operational area in the days preceding the launch, the dry lake bed located to the west of the FAR Rocket Range was covered with roughly a foot (0.3 m) of water. Pushed primarily by the high-altitude westerly winds, the rocket drifted into the center of the lake bed and landed in approximately a foot (0.3 m) of high-salinity standing water. The fine granular sand of which the bed of the lake is composed, made recovery on foot impractical, so a kayak was used to retrieve the rocket's components. Figure 80 shows an aerial view of the rocket's recovery. In the image, the rocket's nose cone section, drogue parachute, and the initial impact points of the components are annotated. In addition, the blue kayak used for retrieval and the author are shown in the left-hand portion of the image.

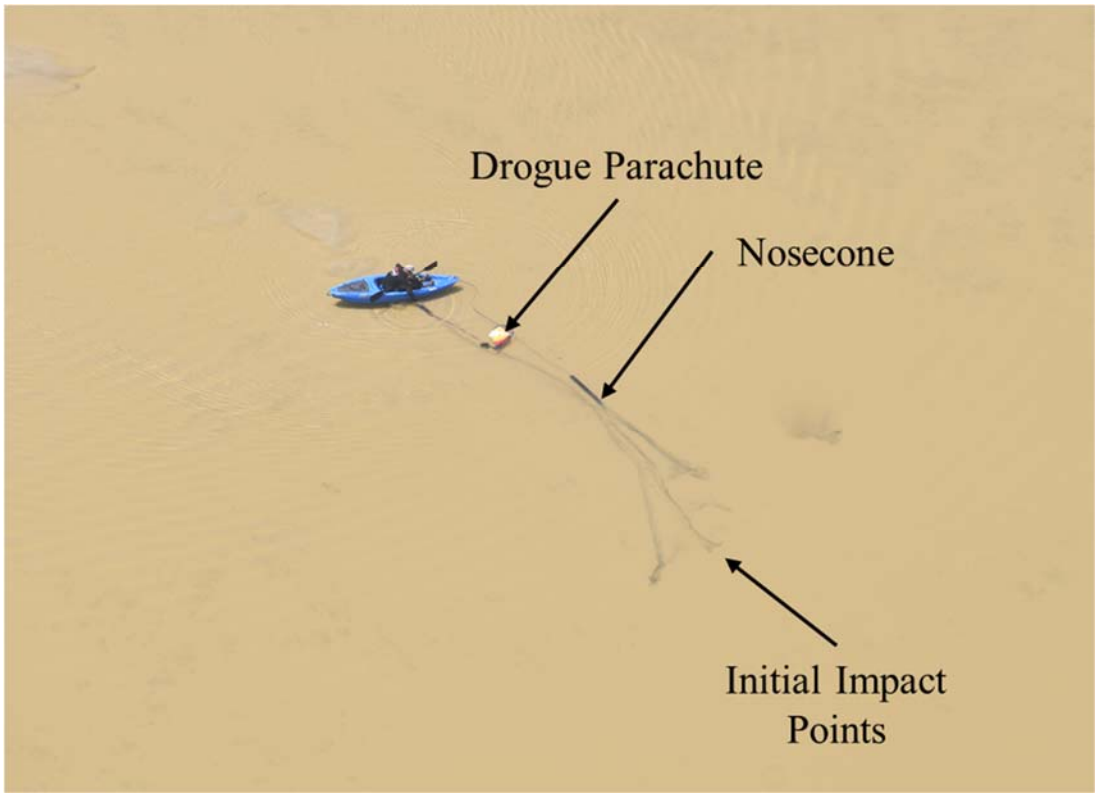


Figure 80. Annotated Rocket Recovery Photo. Adapted from [78].

Upon retrieving the components, it was discovered that the lower airframe of the rocket with the motor casing and fin can had detached from the drogue parachute shock cord. After a preliminary search of the landing area, the lower airframe section was unable to be found and was subsequently annotated as a loss. In attempts to recover the onboard data, the electronic components were rinsed in fresh water and stored in rice for approximately 36 hours.

3. Data Analysis and Lessons Learned

Due to the high salinity of the lake water, the electronic components on the rocket underwent significant corrosion. Figure 81 shows a comparison of a new and the recovered RPi. The image highlights the deteriorated state of the recovered RPi's metal components.

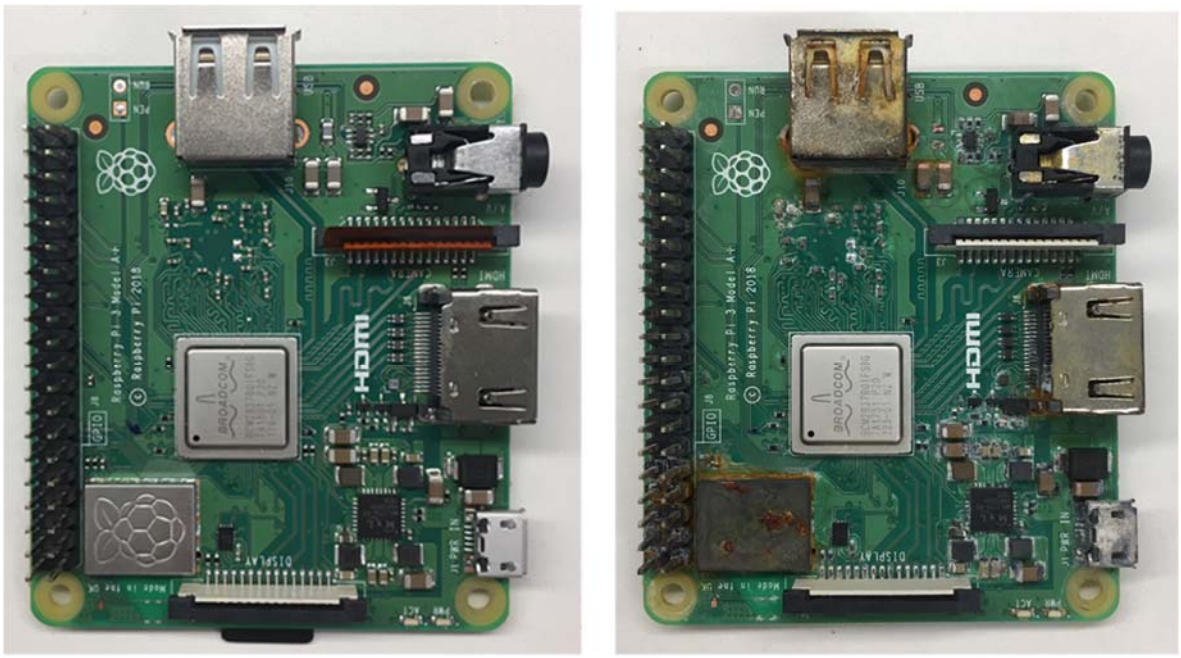


Figure 81. Comparison of New and Recovered RPi

The corrosion of the electronic components caused complete onboard data loss of the TeleMega flight computers and BigRedBee GPS tracker. However, the telemetry data from the TeleMega flight computer was captured and because the custom sensor board was utilizing an SD card for data storage, a full recovery of its data was possible. Table 19 lists the various key flight parameters from the test flight as indicated by the custom sensor and TeleMega telemetry data.

Table 19. Sustainer Test Flight Key Flight Parameters

Maximum Acceleration	Maximum Velocity	Maximum Altitude (AGL)
590 ft./s^2 180 m/s^2 18 Gs	2,800 fps 850 m/s Mach 2.5	39,000 ft. 11,800 m

Figure 82 displays the acceleration data from the custom sensor board IMU. The figure highlights the various events and stages of the rocket's flight. Of note in the image,

the y-axis of the data (depicted in orange) is oriented along the direction of travel of the rocket. Though the launch acceleration depicted in the figure is saturated at -8 Gs, cross-correlation of the IMU data with the high-g accelerometer data shown in Figure 83 indicates that the rocket achieved a maximum acceleration of 18 Gs.

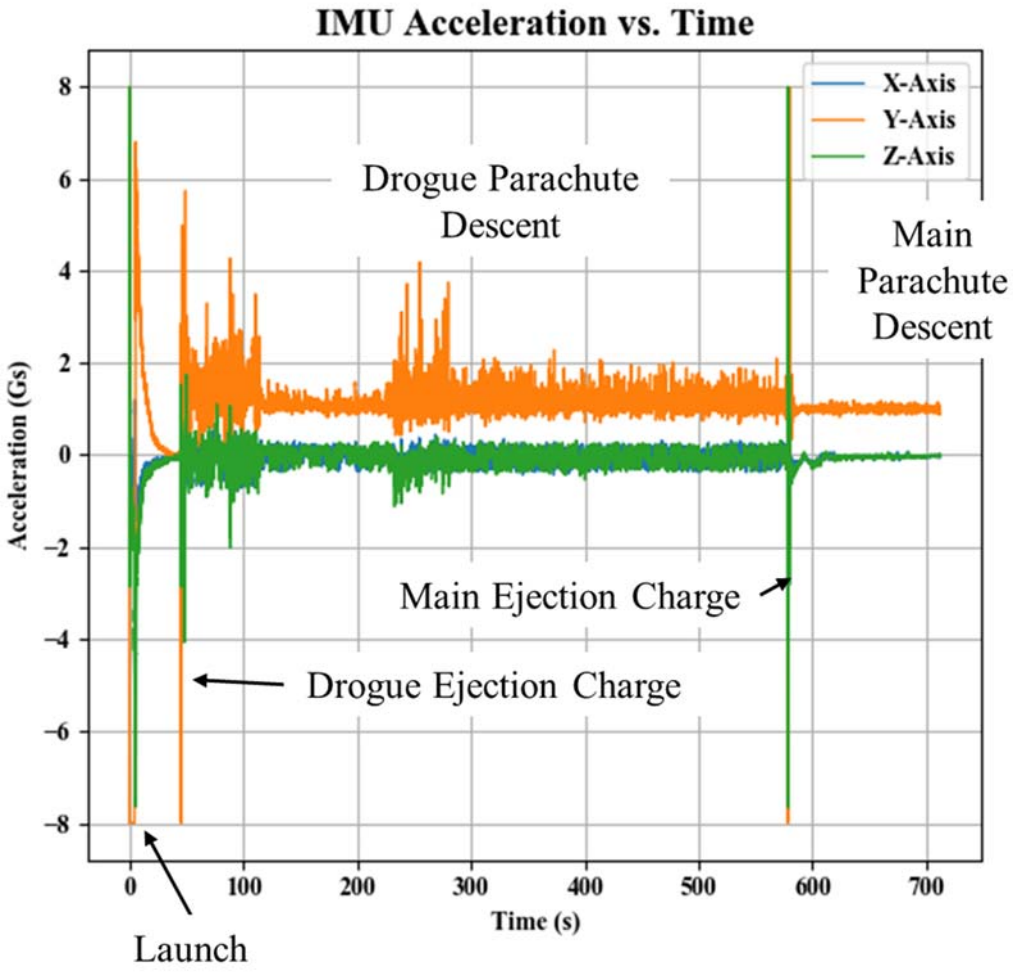


Figure 82. Sustainer Test Flight IMU Acceleration Data

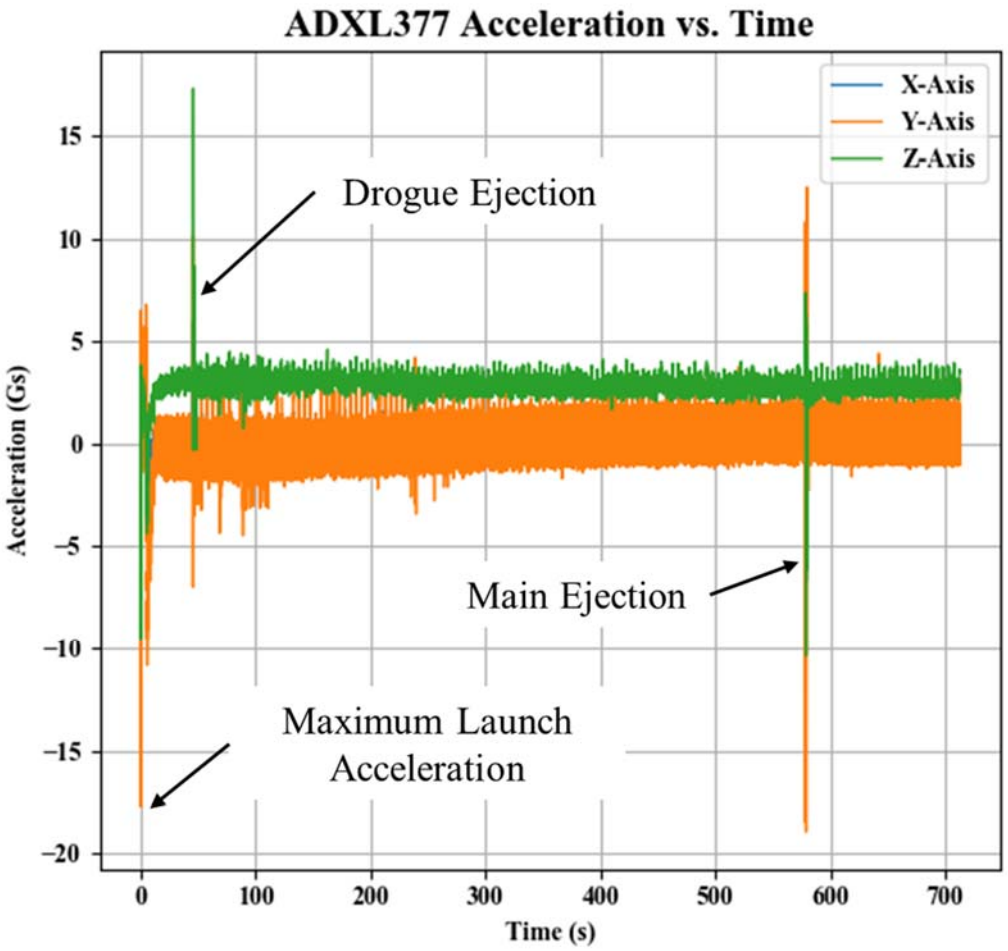


Figure 83. Sustainer Test Flight High-G Accelerometer Data

Figure 84 displays the gyroscope data from the custom sensor board IMU. As with the accelerometer data, the y-axis of the data (depicted in orange) is oriented along the direction of travel of the rocket. Examination of the image reveals the rocket was rotating at nearly 280 revolutions per minute during its ascent. In addition, the image reveals the potential point of the lower airframe detachment from the rest of the rocket components. It can be seen in the figure that at 234 seconds into the rocket's flight, the descent under the drogue parachute becomes increasingly unstable from the previous 100 seconds of descent. Cross-correlation of this point with onboard video footage reveals a dull popping sound that can be heard in the footage audio. Though the video is not angled to provide visual confirmation that this time corresponds to the lower airframe detaching

from the shock cord, it can be reasonably assessed that this is the moment of detachment given the auditory evidence, gyroscope data deviation, and fact that the lower airframe is never seen again in the flight footage after this moment.

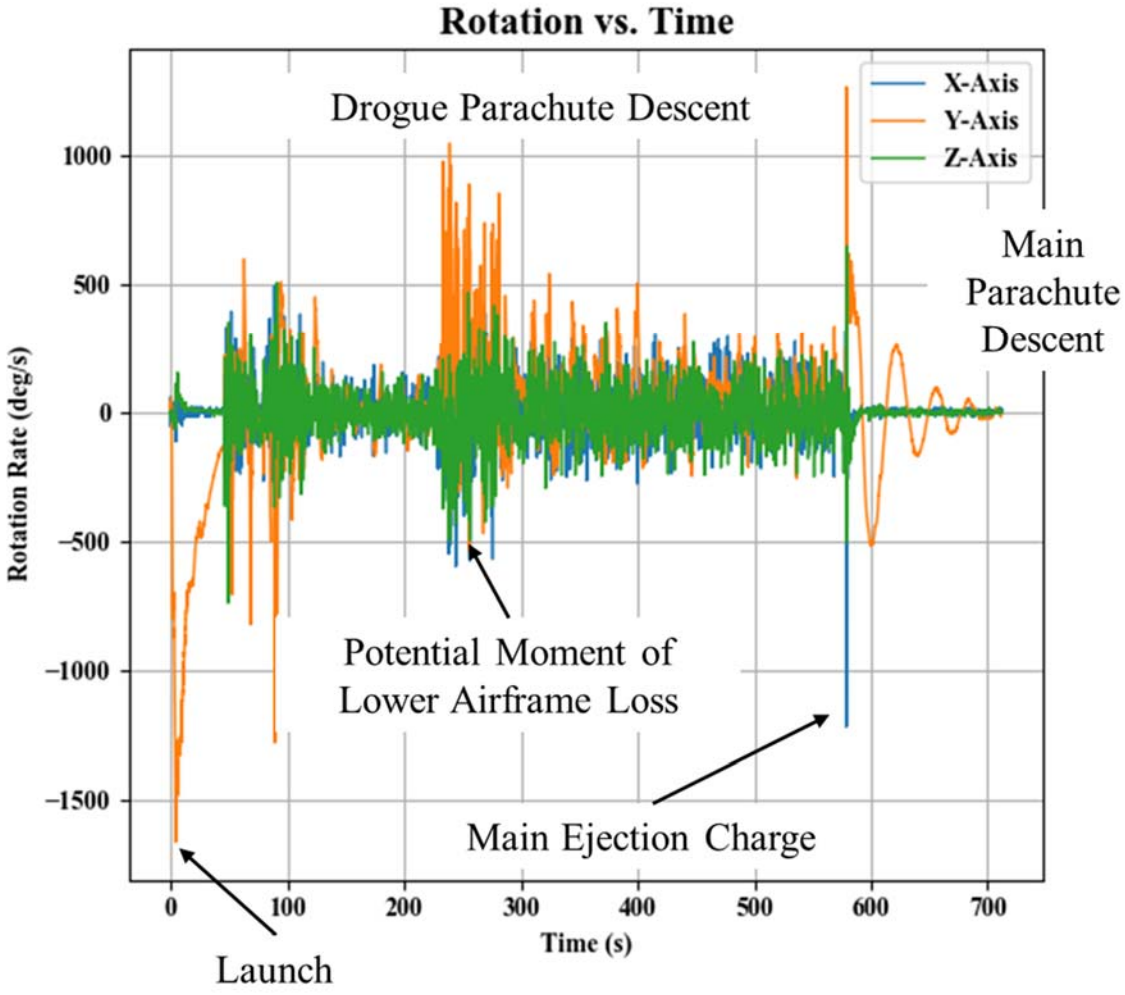


Figure 84. Sustainer Test Flight IMU Gyroscope Data

Based on the flight path, landing location, time of lower airframe loss, and limited GPS data from the TeleMega telemetry stream, it is estimated that the lower airframe is at a location of 35°20’50.6”N 117°51’43.3”W. Figure 85 is a 2D KML rendering of the limited GPS data captured from the TeleMega telemetry stream. In the image the

different phases of the rocket's flight are annotated. In addition, the possible location of the lower airframe is marked.

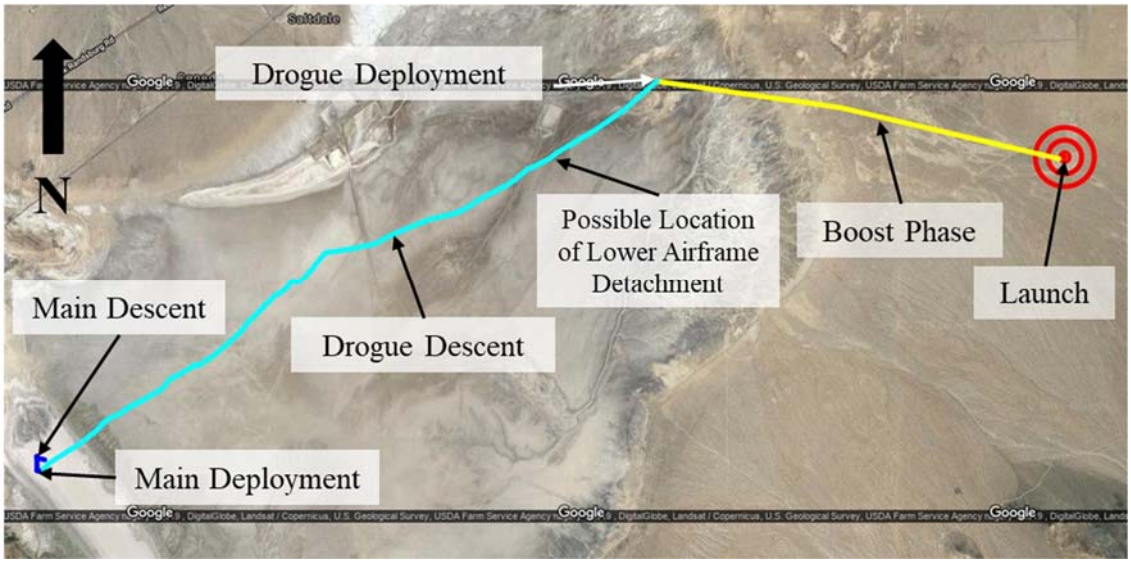


Figure 85. 2D KML Rendering of TeleMega GPS Telemetry Data

Reasons for the lower airframe detachment can be seen in the design of the rocket's motor retention device. Figure 86 is a picture of the minimum-diameter motor retention device used in the rocket. The aluminum device is epoxied to the inside of the rocket's airframe and used to retain the rocket's motor and serves as an attachment point for the rocket's drogue parachute to the lower airframe of the rocket. In the image, the shock cord attachment point is highlighted. The 5/16 in. (0.79 cm) steel eyebolt screws into the aluminum retention device and serves as the shock cord attachment point.

Figure 86. Motor Retention Device

Upon recovery of the rocket, it was discovered that the steel eyebolt had become unscrewed from the motor retention device. Figure 87 shows the hardware of the drogue parachute attachment as it was recovered. In the image the steel eyebolt, quick-link connector, and drogue parachute shock cord are pictured. It can be seen from this image that the steel eyebolt is still connected to the rest of the hardware and has not been damaged. This indicates the eyebolt became unscrewed from the motor retention device, thus separating the lower airframe of the rocket from the drogue parachute.

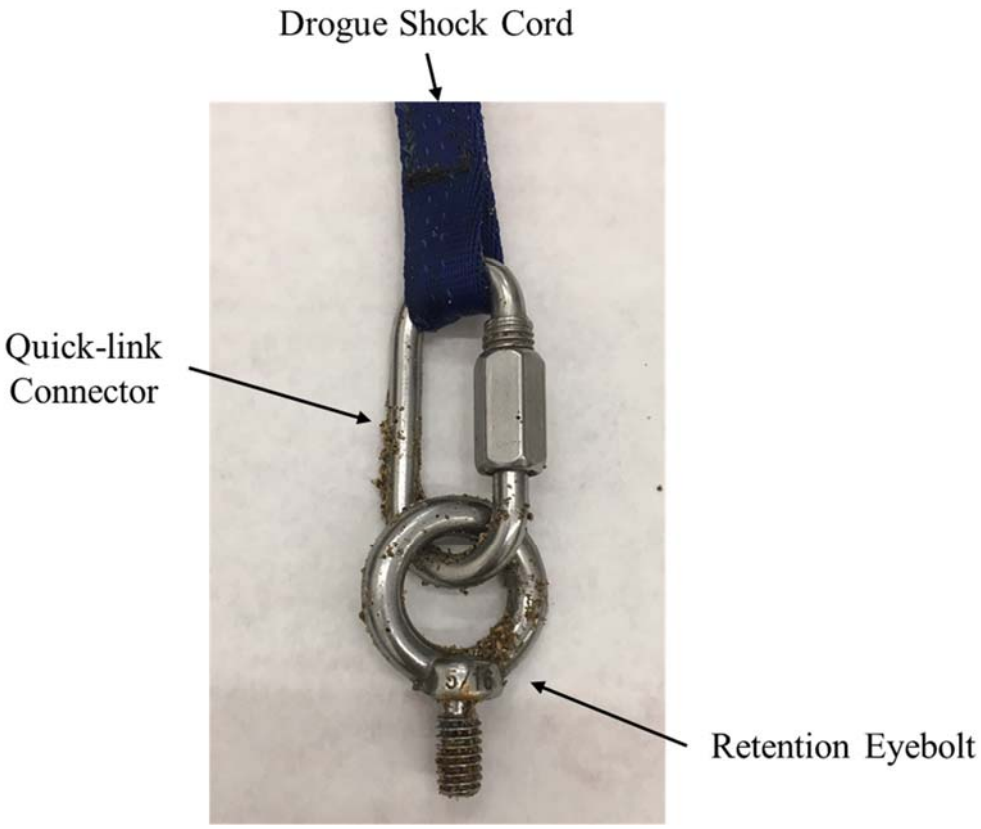


Figure 87. Drogue Parachute Shock Cord Motor Retention Device

Figure 88 is a screen capture of onboard video that shows the lower airframe of the rocket in early stages of drogue parachute descent when it was still connected to the rest of the rocket. The image appears to show that the drogue parachute shock cord was significantly twisted in a direction that would apply force towards unscrewing the bolt. The image further confirms that the bolt simply became unscrewed and was the reason for the lower airframe detachment. Though the bolt was hand tightened as part of the prelaunch checklist, it should be epoxied or more permanently secured in place to mitigate this issue for future flights.

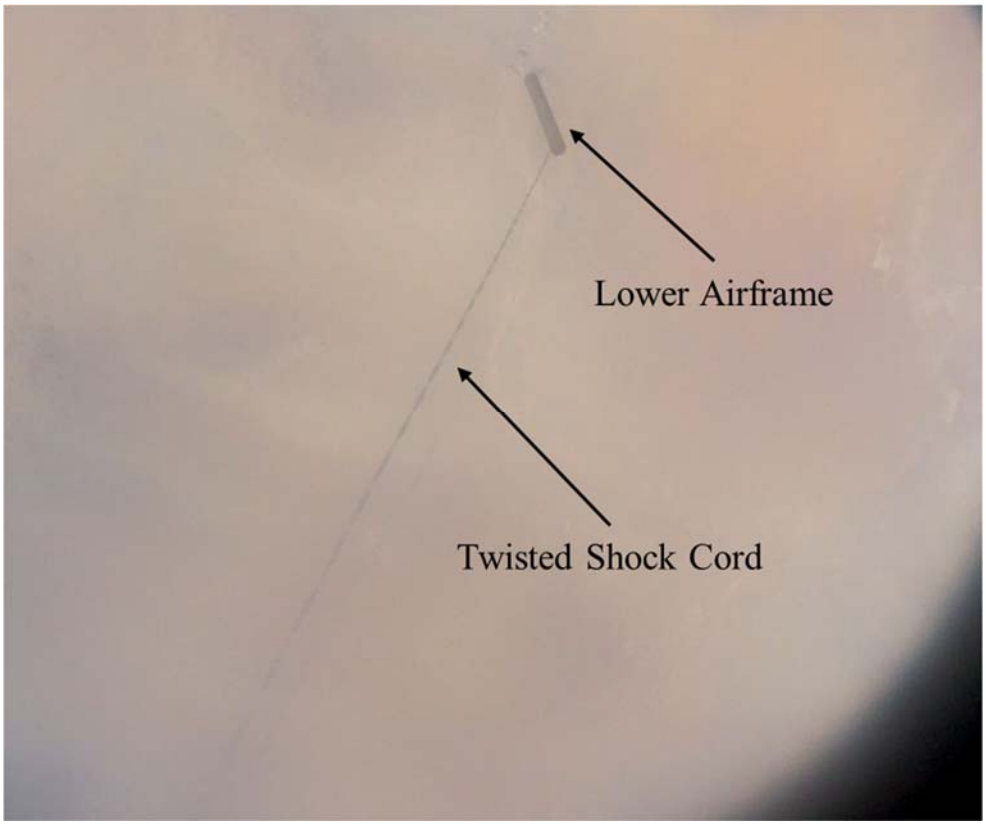


Figure 88. Screen Capture of Onboard Video Revealing Twisted Shock Cord

4. Flight 6 Lessons Learned

Several lessons were learned from the successful flight and partial recovery of flight 6:

1. Physical confirmation of the CG and CP locations is critical to ensuring rocket stability. Final confirmation should be done with a balancing jig once all the rocket components are assembled and prior to leaving the hangar. Ballast devices such as sand in small Ziploc bags should be on standby should the CG location need to be shifted prior to launch. Confirmation of the CG location should be captured and properly documented as part of the pre-launch checklist.

2. All bolts and screws need to be secured with back out prevention devices. Depending on the bolt or screw location and functionality, such methods for preventing back out include: the use of Loctite or epoxy, safety-wire, and jam or lock nuts.

3. All flights should include a data capturing device that stores data on an SD card. Such a method for data storage has proven reliable for data recovery in the event that the flight sensor becomes unusable (as seen in this flight with the corroded sensors). This way the SD card can be transferred from the unusable sensor and the data can be recovered.

5. Flight 6 Conclusions

The redesign of the NPS SSAG high-power rocket sustainer stage functioned as designed and met the objectives of the test launch. Though the malfunction of the eyebolt and water landing negated the reusability of the test rocket, the events do not detract from the functionality of the design. It is believed that by better securing the bolt in place for future flights and better judging the launch angle for high-altitude winds, these two issues can be mitigated.

Though there was no additional payload on board, the rocket effectively demonstrated the capability to carry technical and military payloads to significant altitudes. By forgoing the incorporation of the CubeSat form factor from the previous design, the new sustainer section proved to be more robust and able to withstand the relatively severe forces associated with low-altitude transonic flight. In future development of the program and iterations of the rocket's design, the NPS SSAG high-power rocket team will have to decide whether to try to return to accommodating a CubeSat form factor using a redesigned transition with greater sectional overlap or by increasing the diameter of the rocket's sustainer airframe or continue with the new payload form factor.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

The minimum requirements for the operation of a sub-orbital rocket research and test program within the NPS SSAG have been established. The program has demonstrated the potential to complement the capabilities of the well-established HAB program and produce similar technical and military research benefits to the NSRP and Air Force SRP on a budget and scale within the operational and financial capabilities of the NPS SSAG.

The successful test flight (flight 6) of the rocket's sustainer stage indicate the design for the preliminary launch vehicle is capable of serving as the foundational test platform for the NPS SSAG HPRP. A full system test flight of the booster and sustainer sections of the rocket will be necessary for achieving the full system capabilities of the rocket.

1. Program Operation

To maintain the HPRP within the NPS SSAG, it is recommended that a minimum of two SSAG staff members become QUAL/CERT and amateur high-powered rocketry level 3 certified. By maintaining qualified staff members, the Navy explosive handling requirements can be fulfilled and high-powered rocketry institutional knowledge is maintained within the SSAG. Subsequently, students (who have a relatively short research timeline) will not have to spend time acquiring the administrative certifications for conducting high-powered rocketry operations. Instead, students can focus on developing and testing new rocket and payload technologies with the support of the SSAG staff.

2. Program Cost

In addition to the material cost of each rocket, it is estimated that the operational cost of the NPS SSAG HPRP will be approximately $45,000 annually. This figure

includes travel reimbursement, labor, launch fees, and a 10% margin for a relatively aggressive launch schedule of one launch per quarter.[5]

B. FUTURE WORK

The establishment, continued operations, and expansion of the NPS SSAG HPRP is a large undertaking with a multitude of areas for continued work. From rocket subsystem design improvements to the review of the program's operational procedures, the areas of future work span a wide variety of topics. The following is a brief list of some of the more immediate topics for future work.

1. Booster Analysis and Subsequent Flight

Due to time restrictions, a booster flight was not completed. As the program and rocket currently stand, the design of the booster section of the rocket is ready for construction. It is suggested the design be reexamined to see whether lessons learned from the initial sustainer flight (flight 5) can be incorporated into the booster section as well (e.g., fiberglass as the airframe material and the incorporation of an aluminum fin can). Additionally, for the flight to be conducted at the FAR Rocket Range, a Class-3 High Power Rocket Waiver must be filed and approved by the FAA. Details associated with the requirements of the waiver can be found in [79].

2. Liquid Rocket Engine

Much of the operational burden on the NPS SSAG HPRP is due to the explosive handling requirements necessary for the handling and use of solid rocket propellant. Though more complex and difficult to integrate, a liquid-fueled rocket engine would reduce the associated explosive handling burden and simplify the program's operational and logistical considerations. Additionally, a liquid-fueled rocket engine would allow for rocket flight profile features that are unachievable with a solid propellant motor. One such example is the throttling capability of a liquid engine. Unlike a solid propellant

[5] Labor hours are estimated at a rate of $100/hour for an average of five hours per week. Travel includes reimbursement for five individuals and the launch fee is estimated at an average of $1,000 per day.

motor, a liquid engine's thrust can be throttled. One potential benefit of this capability is reducing the acceleration of the rocket as it approaches the point of maximum dynamic pressure in its flight profile. Figure 89 is a plot of the aerodynamic pressure profile generated during pre-flight analysis for flight 5. It can be seen from the image that the dynamic pressure profile is relatively steep and the point of maximum dynamic pressure occurs early in the flight. By throttling the engine, the relatively steep dynamic pressure curve can be smoothed and the point of maximum dynamic pressure can be shifted to the right on both plots. This reduces aerodynamic forces on the launch vehicle, reduces the mission risk, and allows for optimization of the airframe structure weight. Development of a liquid rocket engine is a logical next step for the program and will enable more dynamic rocket research missions.

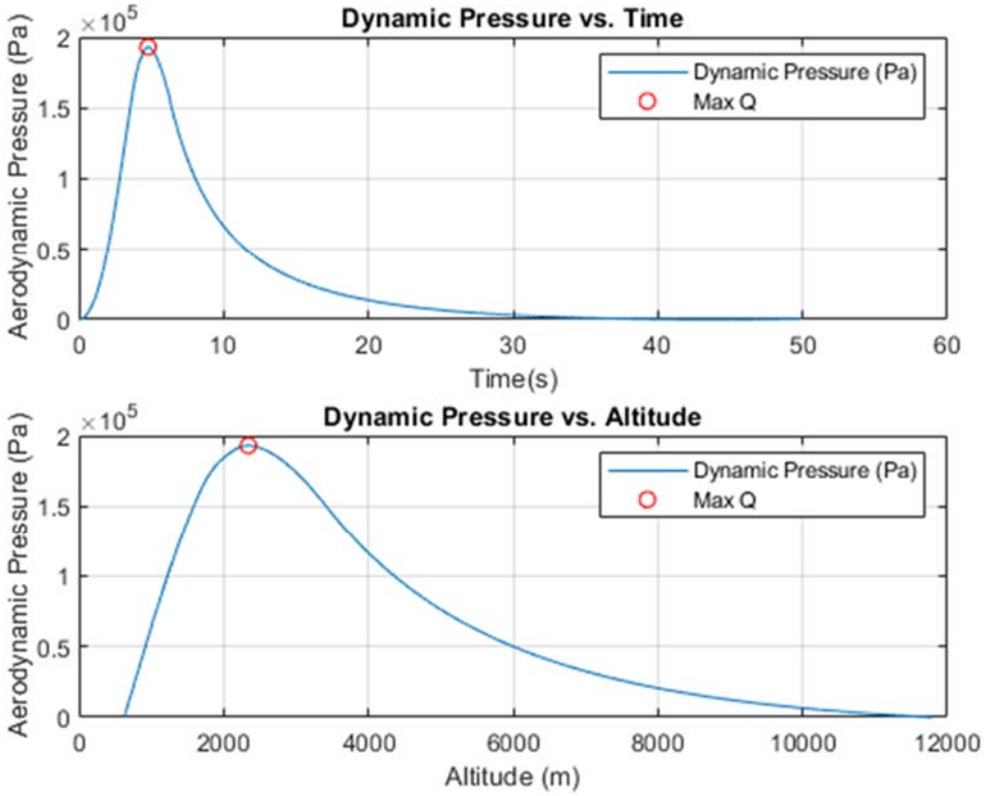


Figure 89. Flight 6 Dynamic Pressure as a Function of Time and Altitude

3. NPS SSAG HPRP Flight Computer

Though the custom sensor board developed for this thesis currently only functions as an onboard data capturing device, it was designed with the ultimate goal of becoming the rocket's main flight computer. With the addition of some software logic and GPIO pin breakouts, the board is capable of achieving that goal. Development of a custom flight computer would provide for complete flexibility of the rocket's mission profile. In addition, if control systems hardware is added to the rocket, dynamic rocket stability and thrust vectoring is a possibility. Such features would drastically increase the capability of the rocket and its potential to support technical and military research. The following is a brief summary of the lessons learned in the design and testing of the current custom sensor board:

1. PCB solder pads should be oversized. This makes hand soldering small surface mount components easier. Additionally, outsourcing component placement can be a time saving alternative to hand soldering.

2. Components that do not have access to their pads from the side (e.g., the LSM9DS1 and ADXL377) must be placed on the board first and connected via reflow soldering. These components need to then be tested for functionality prior to attaching any other components. That way if there is a bad connection, it can be detected and the reflow soldering process can be attempted again.

3. The LSM9DS1 IMU does not contain software that produces absolute orientation. It is recommended that an alternate IMU, such as the Bosch BNO055 which does contain such software for fusing multiple sensor inputs, replace the LSM9DS1 as the primary IMU. Doing so would reduce the software programming burden needed for absolute orientation measurements for future iterations of the flight computer.

4. The MHX radio can interfere with the GPS signal and prevent GPS signal lock. To mitigate this as a possibility during flight, it is recommended that

a high gain active GPS antenna be used and placed as far away from the MHX as possible.

5. As seen in Figure 83, the ADXL377 accelerometer produces relatively noisy data. It is recommended that the acceleration data from the ADXL377 be filtered if used for real-time logic decisions.

6. The fixed voltage LM1085 linear regulators do not always produce the advertised output voltage. This can result in a low-voltage being supplied to the RPi and can subsequently brownout the device. To remedy this issue, it is recommend that the adjustable version of the LM1085 be incorporated into the board as its output voltage is not fixed.

7. The MPL3115A2 barometer minimum data acquisition rate is one second. It is recommended that a faster sampling barometer be integrated for real-time altitude logic decisions.

4. Curriculum Incorporation

The NPS SSAG HPRP provides many hands-on educational benefits that are hard to achieve in a classroom setting. As seen in this thesis, amateur rocketry incorporates a wide range of academically-challenging topics. By working through real-world challenges that are associated with each of these topics, students acquire and retain a wide array of general knowledge and skills. By incorporating lectures and labs for the Space Engineering and Space Operations curriculums that use the NPS SSAG high-power rocket, this knowledge and these skills will be further enhanced. Such a concept serves to support the ultimate goal of producing agile-thinking military officers who are able to apply well-retained knowledge to new and dynamic technical situations.

5. Milestone Review Procedures and Details

The last topic for discussion for future work is establishing the detailed procedures associated with each milestone review. By generating specific due outs and timelines for each milestone review, stakeholders can better track the mission's progress

and facilitate its success. Additionally, standardization of the procedures associated with the conduct of each review will decrease mission briefing ambiguity and provide for better communication. Ultimately, these procedures will make the program more efficient and streamline the mission timeline. Such effects will decrease the mission development time and increase the program's mission success rate. In turn, this will allow for more launches and better results from the program.

APPENDIX A. MATLAB CODE

A. ROCKET FLIGHT SIMULATION ESTIMATE – FULL MISSION

Thesis Rocket

The program Thesis Rocket is designed to calculate key flight performance parameters of the NPS Class-3 High Power Rocket. Motor thrust data provided by thrustcurve.com for the O3400 sustainer motor and Robert DeHate of Animal Motor Works for the Q15782 booster motor. The example output for this specific script shows the estimated key flight performance parameters for the two-stage configuration using the initial sustainer (flight 5) and booster design.

Source Author: Dillon Pierce
Name of File: ThesisRocket.m
File Location: ssagcommon$\High Power Rocket\MATLAB
Date Last Modified: 23 May 2019

Inputs: Various rocket parameters
Outputs: Rocket key flight performance parameters

Program Start

Program start will clean the command window, workspace variables, and format MATLAB output in a compact mode

```
clear, clc, format compact;
```

Given Rocket Characteristics and Other Inputs

Defined Rocket Characteristics

```
global rhos g0

[booster_datafile,~] = uigetfile({'*.csv'},'Select Booster Thrust Data File');     %
Select Booster Thrust Data
booster_thrust_array = xlsread(booster_datafile);      % Defining the array that contains
the data
[sustainer_datafile,~] = uigetfile({'*.csv'},'Select Sustainer Thrust Data File');  %
Select Sustainer Thrust Data
sustainer_thrust_array = xlsread(sustainer_datafile);      % Defining the array that
contains the data

d_booster = .1524;              % Body Tube Diameter of Booster (m)
d_sustainer = .1016;            % Body Tube Diameter of Sustainer (m)
Cd = 0.7;                       % Coefficient of Drag based on Von Karman nose cone (.516
from test data)
m_booster = 10;                 % Booster Empty Mass (kg) (22 lbs)
m_sustainer = 12;               % Sustainer Empty Mass (kg) (24 lbs)
m_tot = m_booster+m_sustainer;  % Total Rocket mass (kg)
```

```
m_Q15782_prop = 40.75;          % Booster Motor Propellant Mass (kg) (90 lbs)
m_Q15782_tot = 49.84;           % Booster Motor Total Mass (kg) (110 lbs)
m_O3400_tot = 16.84;            % Sustainer Motor Total Mass (kg) (37 lbs)
m_O3400_prop = 10.93;           % Sustainer Motor Propellant Mass (kg)(24 lbs)

A_boost = pi*(d_booster*.5)^2;  % Cross-sectional Area of booster (m^2)
A_sust = pi*(d_sustainer*.5)^2; % Cross-sectional Area of sustainer (m^2)

fidelity = 1000;    % Fidelity increment for calculations
launch_alt = 626;   % Launch site altitude (m)
stage_delay = 17;   % Sustainer Stage ignition delay from launch (s)
sep_time = 17;      % Estimated time of stage seperation (s)
```

Constants Assumed

```
rhos = 1.225;       % Density of air in kg/m^3 at 20 deg. Centigrade
g0 = 9.807;         % Gravity constant (m/s^2)
```

Booster Thrust Phase

```
% Calculate Thrust Array from Excel Spreadsheet
t_boost = linspace(booster_thrust_array(1,1),booster_thrust_array(end,1),fidelity);   %
Boost phase time
tdata = booster_thrust_array(1:end,1);                        % Time Data from excel
spreadsheet (s)
Tdata = booster_thrust_array(1:end,2);                        % Thrust Data from excel
spreadsheet (N)
T_boost = interp1(tdata,Tdata,t_boost,'linear');    % Interpolation to get thrust (N) at
each time increment


% Calculate Mass Array of rocket during booster thrust phase
mint = m_tot+m_Q15782_tot+m_O3400_tot;               % Initial Mass (kg)  (Empty Rocket +
Motors)
mfin = m_tot+m_O3400_tot+m_Q15782_tot-m_Q15782_prop; % Final Mass (kg) (Empty Rocket +
sustainer motor - empty booster)
dmdt = (mint-mfin)/t_boost(end);  % Linear Mass change over burnout time
mass_boostphase = -t_boost.*dmdt+mint;
mass = mass_boostphase;


% Preset Allocations
dt = t_boost(2)-t_boost(1);         % Time increment
v = zeros(1,length(t_boost));       % Empty set pre-allocation for velocity
a = zeros(1,length(t_boost));       % Empty set pre-allocation for acceleration
h = zeros(1,length(t_boost));       % Empty set pre-allocation for height
rho = zeros(1,length(t_boost));     % Empty set pre-allocation for Density
D = zeros(1,length(t_boost));       % Empty set pre-allocation for Drag
v(1) = 0;                           % Define v(1)
h(1) = launch_alt;                  % Define h(1) (Launch Site Altitude (m))
a(1) = 0;                           % Define a(1)
```

```
rho(1) = rhos*exp((-1/7800)*h(1)); % Define rho(1)


for k = 1:length(t_boost)-1
    rho(k+1) = Density(h(k));                    % Atmoshperic Density
    D(k+1) = Drag(v(k),h(k),A_boost,Cd);         % Function for drag (N)
    a(k+1) = acceleration(T_boost(k),D(k),mass_boostphase(k)); % Acceleration function to
find dvdt
    v(k+1) = v(k) + a(k)*dt;                     % Euler's Method for ODE's to get
velocity
    h(k+1) = h(k) + v(k)*dt;                     % Euler's Method for ODE's to get height
end
```

Coast phase

```
t_coast1 = linspace(t_boost(end),stage_delay,fidelity); % Coast Time
t = horzcat(t_boost,t_coast1);

for k = length(t_boost):length(t_boost)+length(t_coast1)-1
    if t_coast1(k-length(t_boost)+1) > sep_time
        A = A_sust;
        Cd = Cd;
        m = m_sustainer + m_O3400_tot;
    else
        A = A_boost;
        Cd = Cd;
        m = mfin;
    end
    mass(k+1) = m;
    rho(k+1) = Density(h(k));                    % Atmoshperic Density
    D(k+1) = Drag(v(k),h(k),A,Cd);               % Function for drag (N)
    a(k+1) = acceleration(0,D(k),m);             % Acceleration function to find dvdt
    v(k+1) = v(k) + a(k)*dt;                     % Euler's Method for ODE's to get
velocity
    h(k+1) = h(k) + v(k)*dt;                     % Euler's Method for ODE's to get height
end
```

Sustainer Thrust Phase

```
% Calculate Thrust Array from Excel Spreadsheet
t_sust = linspace(sustainer_thrust_array(1,1),sustainer_thrust_array(end,1),fidelity);
% Boost phase time
tdata = sustainer_thrust_array(1:end,1);                        % Time Data from excel
spreadsheet (s)
Tdata = sustainer_thrust_array(1:end,2);                        % Thrust Data from excel
spreadsheet (N)
T_sust = interp1(tdata,Tdata,t_sust,'linear');    % Interpolation to get thrust (N) at
each time increment
t_boost_sust = t_sust+t_coast1(end);
t = horzcat(t,t_boost_sust);
```

```
% Calculate Mass Array of rocket during sustainer thrust phase
mint = m_sustainer + m_O3400_tot;               % Initial Mass (kg)  (Empty stage +
motor (kg))
mfin = m_sustainer + m_O3400_tot - m_O3400_prop; % Final Mass (kg) (Empty stage +
sustainer casing (kg))
dmdt = (mint-mfin)/(t_sust(end));  % Linear Mass change over burnout time
mass_sustboost = -t_sust.*dmdt+mint;
mass = horzcat(mass,mass_sustboost);

dt = t_sust(2)-t_sust(1);     % Time increment

for k = length(t_boost)+length(t_coast1):length(t)-1
    rho(k+1) = Density(h(k));                    % Atmoshperic Density
    D(k+1) = Drag(v(k),h(k),A_sust,Cd);         % Function for drag (N)
    a(k+1) = acceleration(T_sust(k-(fidelity*2)+1),D(k),mass_sustboost(k-(fidelity*2-
1))); % Acceleration function to find dvdt
    v(k+1) = v(k) + a(k)*dt;                     % Euler's Method for ODE's to get
velocity
    h(k+1) = h(k) + v(k)*dt;                     % Euler's Method for ODE's to get height
end
```

Sustainer Coast phase

```
apogee = 150;
t_coast2 = linspace(t(end),apogee,fidelity); % Coast Time
dt = t_coast2(2)-t_coast2(1);
t_coast2 = t_coast2+dt;                      % Coast Time

for k = length(t)-1:length(t)+length(t_coast2)-1
    mass(k+1) = mfin;
    rho(k+1) = Density(h(k));                    % Atmoshperic Density
    D(k+1) = Drag(v(k),h(k),A_sust,Cd);          % Function for drag (N)
    a(k+1) = acceleration(0,D(k),mfin);          % Acceleration function to find dvdt
    v(k+1) = v(k) + a(k)*dt;                     % Euler's Method for ODE's to get
velocity
    h(k+1) = h(k) + v(k)*dt;                     % Euler's Method for ODE's to get height
end

t = horzcat(t,t_coast2);
```

Dynamic Pressure

```
for k = 1:length(v)
    q(k) = .5*rho(k)*v(k)^2;        % Aerodynamic Pressure (Pa)
end
```

Displaying Data

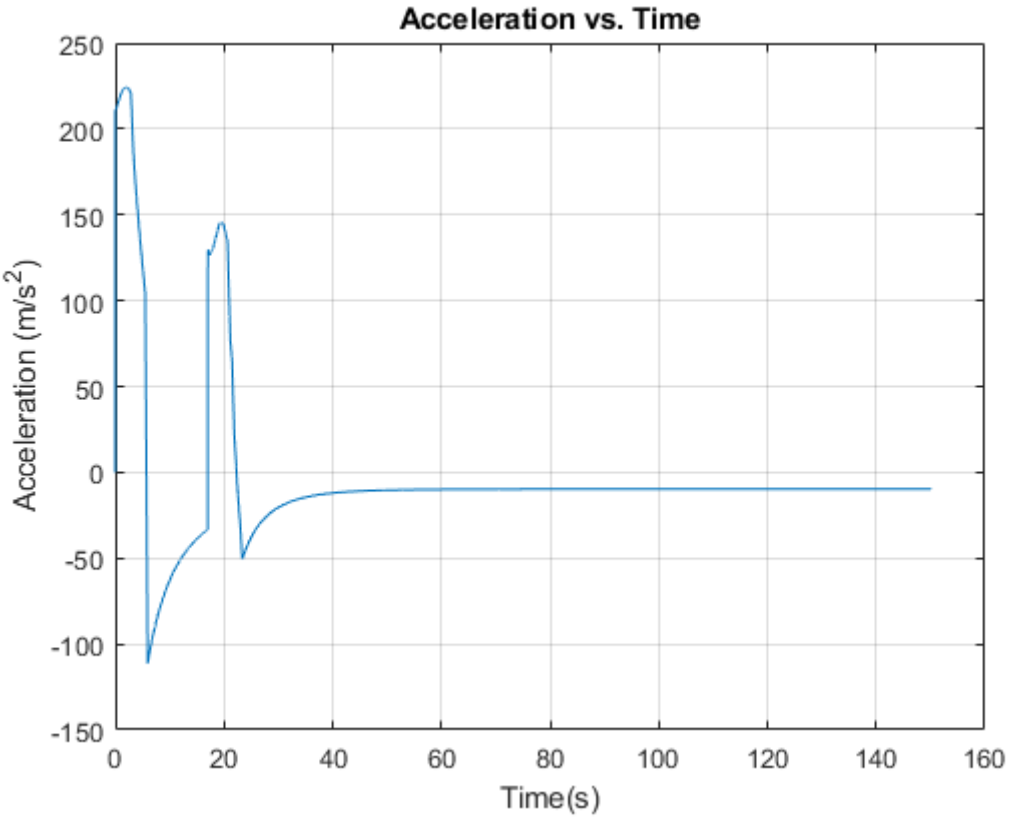
```
[z,y] = max(h);
fprintf('Maximum height is %0.f (m) or %0.f (ft) at %.0f (s)\n',z,z*3.281,t(y))
[z,y] = max(v);
fprintf('Maximum velocity is %0.f (m/s) or %0.f (ft/s) at %.0f (s)\n',z,z*3.281,t(y))
[z,y] = max(a);
fprintf('Maximum acceleration is %0.f (m/s^2) or %0.f (ft/s^2) at %.0f
(s)\n',z,z*3.281,t(y))
fprintf('Velocity at sustainer stage ignition: %0.f (m/s) or %0.f (ft/s) at %.0f
(s)\n',v(2000),v(2000)*3.281,t(y))
[maxq,index_maxq] = max(q);
fprintf('Maximum Dynamic Pressure: %0.f (Pa) at %.0f (s) and %0.f (m-
AGL)\n',maxq,t(index_maxq),h(index_maxq))
```

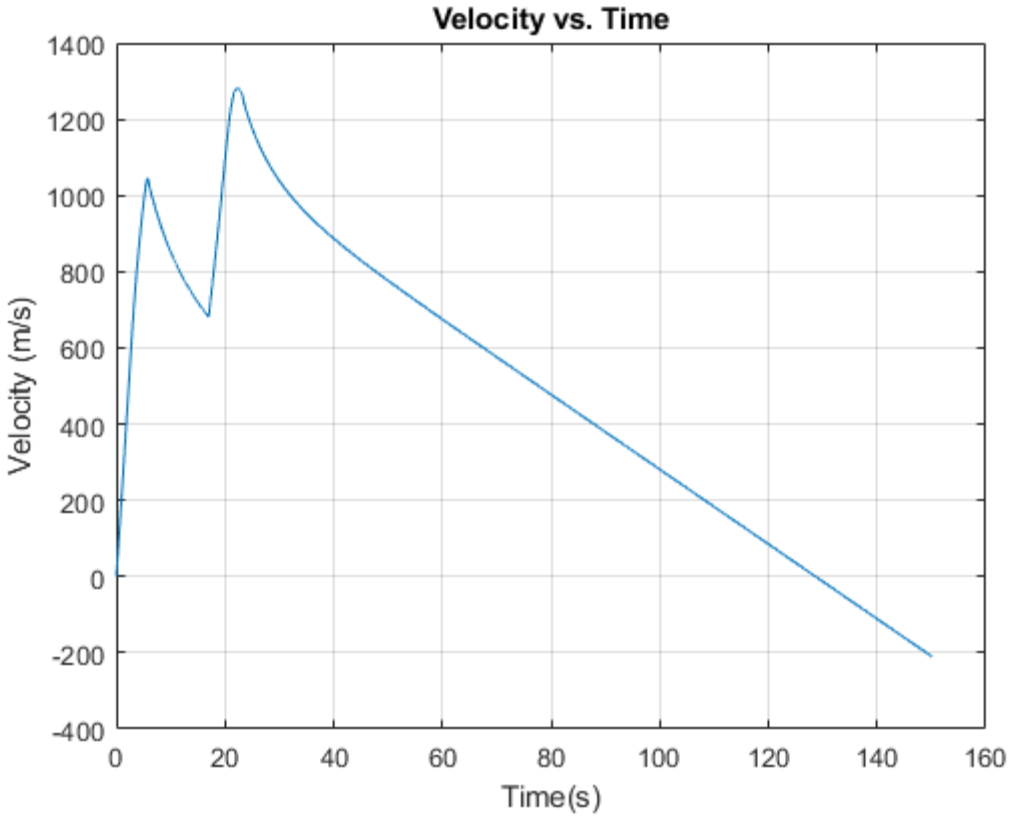
```
Maximum height is 71886 (m) or 235858 (ft) at 129 (s)
Maximum velocity is 1281 (m/s) or 4204 (ft/s) at 22 (s)
Maximum acceleration is 224 (m/s^2) or 736 (ft/s^2) at 2 (s)
Velocity at sustainer stage ignition: 681 (m/s) or 2234 (ft/s) at 2 (s)
Maximum Dynamic Pressure: 405170 (Pa) at 6 (s) and 3856 (m-AGL)
```

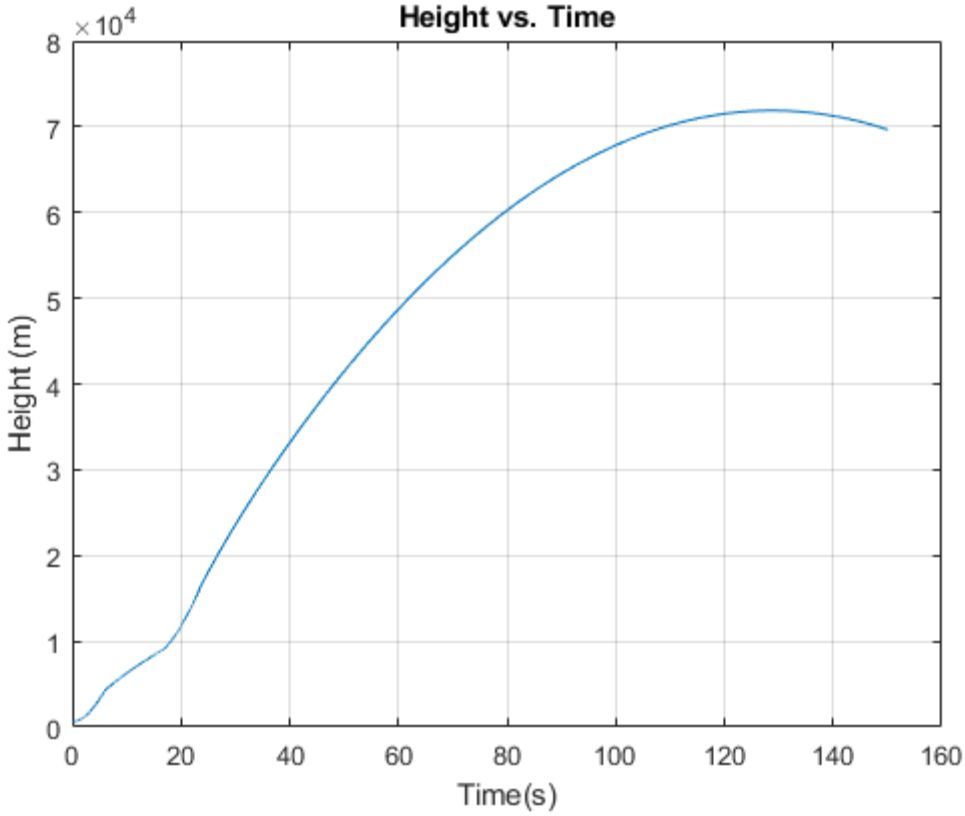
Plotting the Data

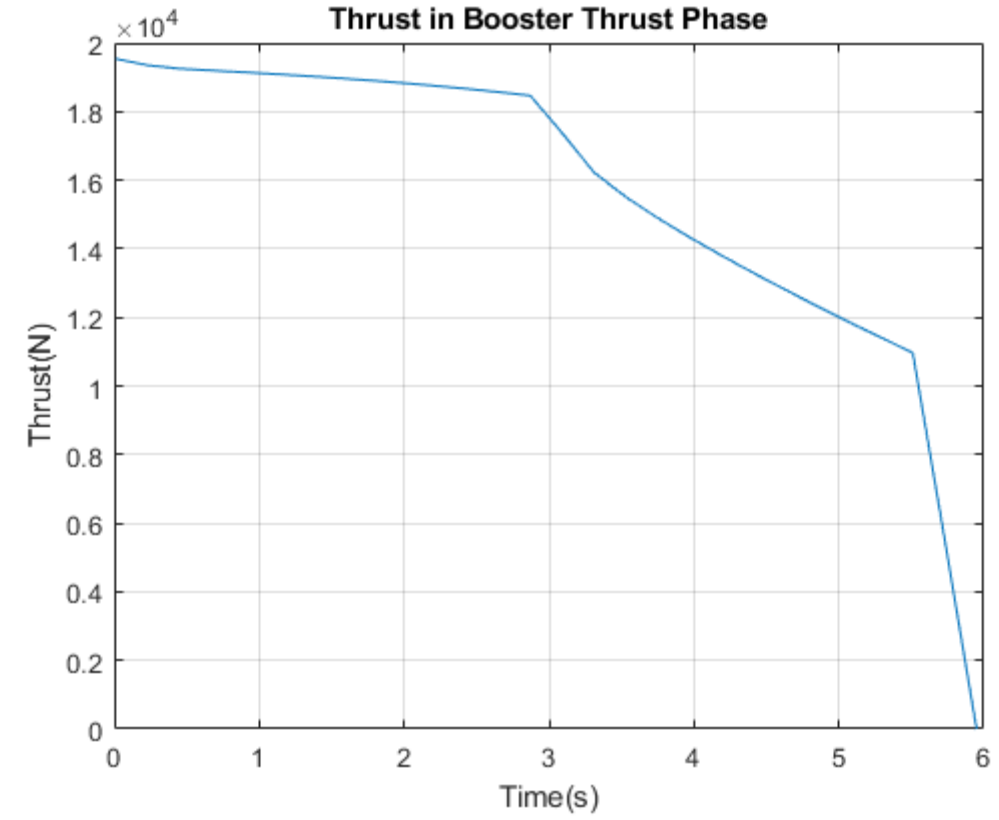
```
figure(1)
plot(t,a), xlabel('Time(s)'), ylabel('Acceleration (m/s^2)'), title('Acceleration vs.
Time')
grid on
figure(2)
plot(t,v), xlabel('Time(s)'), ylabel('Velocity (m/s)'), title('Velocity vs. Time')
grid on
figure(3)
plot(t,h), xlabel('Time(s)'), ylabel('Height (m)'), title('Height vs. Time')
grid on
figure(4)
plot(t_boost,T_boost), xlabel('Time(s)'), ylabel('Thrust(N)'), title('Thrust in Booster
Thrust Phase')
grid on
figure(5)
plot(t_boost_sust,T_sust), xlabel('Time(s)'), ylabel('Thrust(N)'), title('Thrust in
Sustainer Thrust Phase')
grid on
figure(6)
plot(t,mass), xlabel('Time(s)'), ylabel('Mass (kg)'), title('Rocket Mass vs. Time')
grid on
figure(7)
plot(t,rho), xlabel('Time(s)'), ylabel('Air Density (kg/m^3)'), title('Air Density vs.
Time')
grid on
figure(8)
subplot(2,1,1)
plot(t,q,t(index_maxq),maxq,'or'), xlabel('Time(s)'), ylabel('Aerodynamic Pressure
(Pa)'), title('Dynamic Pressure vs. Time')
```

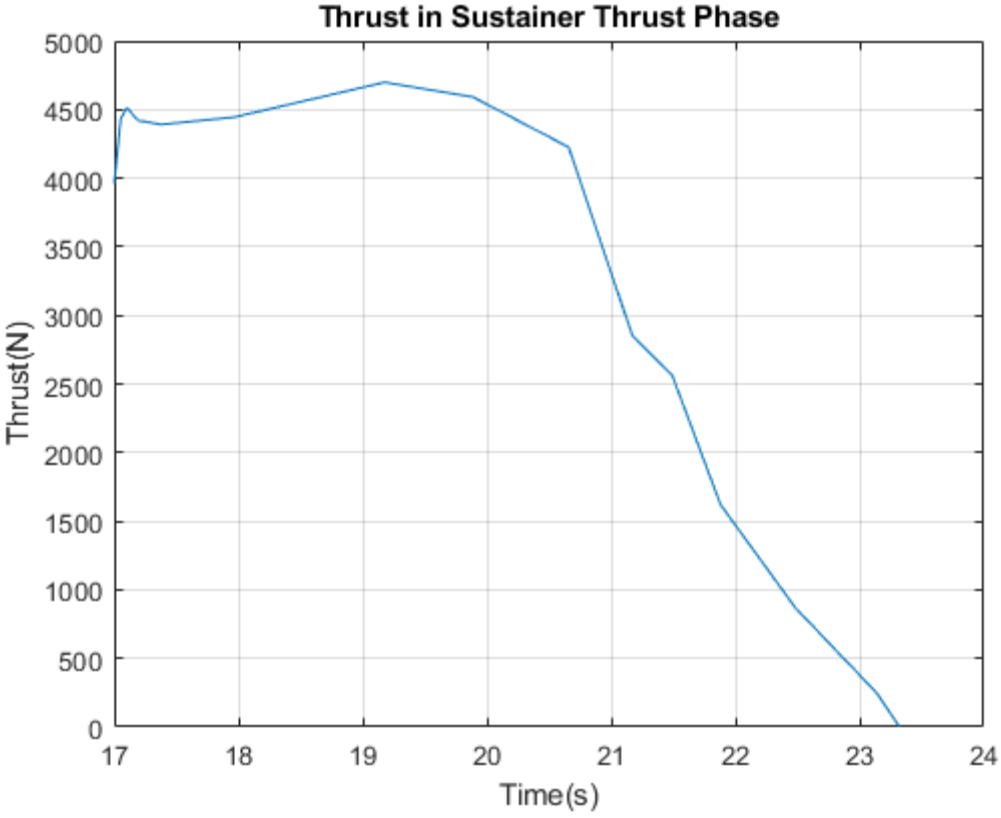
```
legend('Dynamic Pressure (Pa)','Max Q')
grid on
subplot(2,1,2)
plot(h,q,h(index_maxq),maxq,'or'), xlabel('Altitude (m)'), ylabel('Aerodynamic Pressure
(Pa)'), title('Dynamic Pressure vs. Altitude')
legend('Dynamic Pressure (Pa)','Max Q')
grid on
```

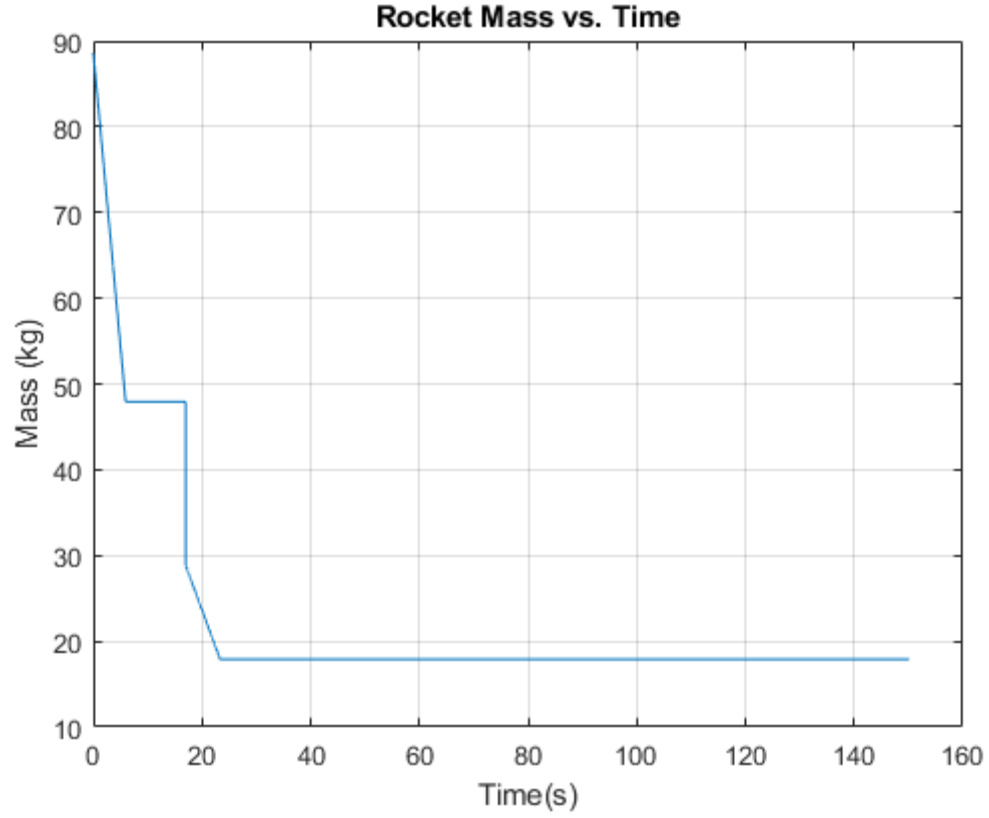


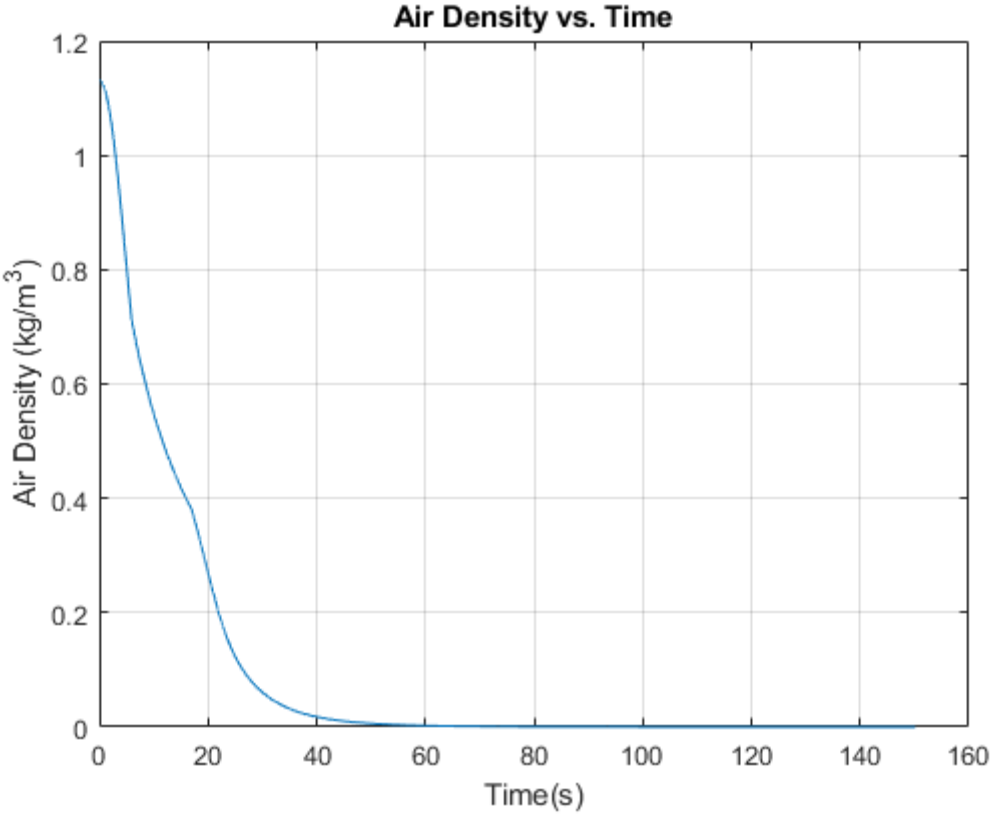

Velocity vs. Time
Velocity (m/s)
Time(s)
1400
1200
1000
800
600
400
200
0
-200
-400
0
20
40
60
80
100
120
140
160

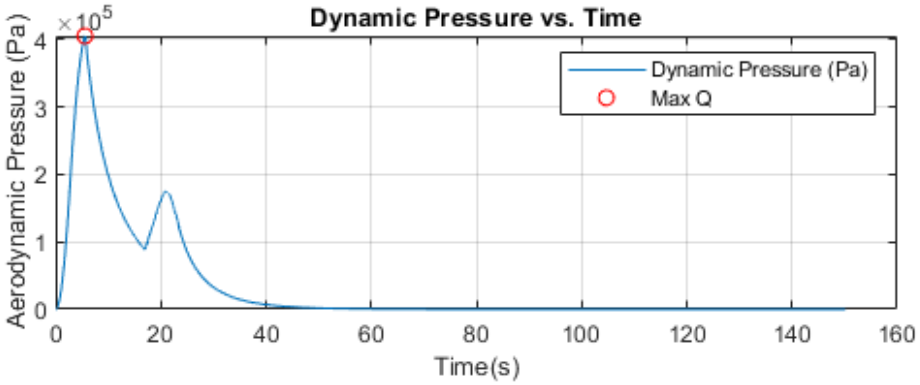

Height vs. Time
×10^4
Height (m)
Time(s)
8
7
6
5
4
3
2
1
0
0
20
40
60
80
100
120
140
160

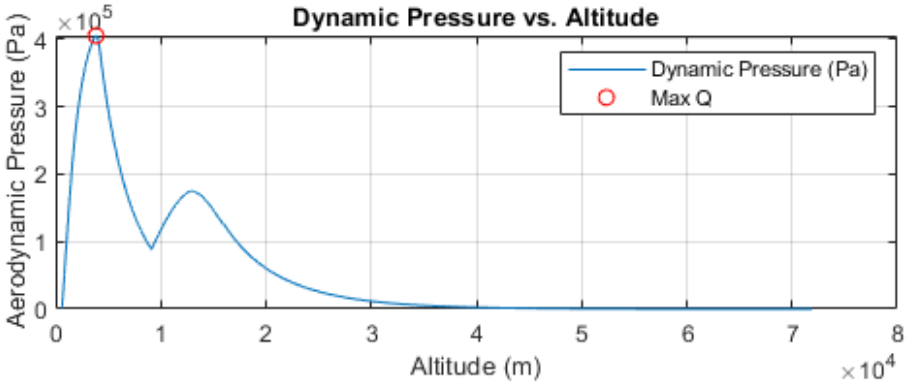

Thrust in Booster Thrust Phase
×10^4
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Thrust(N)
0
1
2
3
4
5
6
Time(s)


Thrust in Sustainer Thrust Phase
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Thrust(N)
17
18
19
20
21
22
23
24
Time(s)


Rocket Mass vs. Time
Mass (kg)
Time(s)
90
80
70
60
50
40
30
20
10
0
20
40
60
80
100
120
140
160


Air Density vs. Time
Air Density (kg/m3)
Time(s)
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
140
160





Functions

```
function [ D ] = Drag( v,h,A,Cd )
%The Drag function calculates the drag on an object at a certain velocity

D = .5*Density(h)*v^2*Cd*A;   % Function for drag (N)


end

function [rho] = Density(h)
% Function for changing atmospheric density
global rhos;

rho = rhos*exp((-1/7800)*h);

end

function [ dvdt ] = acceleration(T,D,M)
% The acceleration function is used to calculate the acceleration of a
% rocket at a given time and velocity.

global g0;

dvdt = (T - D)/M - g0;
end
```

B. ROCKET FLIGHT SIMULATION ESTIMATE – SUSTAINER ONLY

Thesis Rocket – Sustainer Only

The program ThesisRocketSustainerOnly is designed to calculate key flight performance parameters of the NPS Class-3 High Power Rocket (sustainer-only configuration). Motor thrust data provided by thrustcurve.com for the O3400 sustainer motor.

The example output for this specific script shows the estimated key flight performance parameters for the single-stage configuration using the redesigned sustainer (flight 6).

Source Author: Dillon Pierce
Name of File: ThesisRocketSustainerOnly.m
File Location: ssagcommon$\High Power Rocket\MATLAB
Date Last Modified: 23 May 2019

Inputs: Various rocket parameters
Outputs: Rocket key flight performance parameters

Program Start

Program start will clean the command window, workspace variables, and format Matlab output in a compact mode

```
clear, clc, format compact;
```

Given Rocket Characteristics and Other Inputs

Defined Rocket Characteristics

```
global rhos g0

[sustainer_datafile,~] = uigetfile({'*.csv'},'Select Sustainer Thrust Data File');  %
Select Sustainer Thrust Data
sustainer_thrust_array = xlsread(sustainer_datafile);      % Defining the array that
contains the data

d_sustainer = .1016;            % Body Tube Diameter of Sustainer (m)
Cd = 0.7;                       % Coefficient of Drag based on Von Karman nose cone (.516
from test data)
m_sustainer = 12;               % Sustainer Empty Mass (kg) (25 lbs)
m_O3400_tot = 16.84;            % Sustainer Motor Total Mass (kg) (37 lbs)
m_O3400_prop = 10.93;           % Sustainer Motor Propellant Mass (kg)(24 lbs)

A_sust = pi*(d_sustainer*.5)^2; % Cross-sectional Area of sustainer (m^2)

fidelity = 1000;    % Fidelity increment for calculations
launch_alt = 626;   % Launch site altitude (m)
```

Constants Assumed

```
rhos = 1.225;       % Density of air in kg/m^3 at 20 deg. Centigrade
g0 = 9.807;         % Gravity constant (m/s^2)
```

Preset Allocations

```
v(1) = 0;                          % Define v(1)
h(1) = launch_alt;                 % Define h(1) (Launch Site Altitude (m))
a(1) = 0;                          % Define a(1)
rho(1) = rhos*exp((-1/7800)*h(1)); % Define rho(1)
```

Sustainer Thrust Phase

```
% Calculate Thrust Array from Excel Spreadsheet
t_sust = linspace(sustainer_thrust_array(1,1),sustainer_thrust_array(end,1),fidelity);
% Boost phase time
tdata = sustainer_thrust_array(1:end,1);                        % Time Data from excel
spreadsheet (s)
Tdata = sustainer_thrust_array(1:end,2);                        % Thrust Data from excel
spreadsheet (N)
T_sust = interp1(tdata,Tdata,t_sust,'linear');    % Interpolation to get thrust (N) at
each time increment
t_boost_sust = t_sust;
t = t_boost_sust;

% Calculate Mass Array of rocket during sustainer thrust phase
mint = m_sustainer + m_O3400_tot;                % Initial Mass (kg)  (Empty stage +
motor (kg))
mfin = m_sustainer + m_O3400_tot - m_O3400_prop; % Final Mass (kg) (Empty stage +
sustainer casing (kg))
dmdt = (mint-mfin)/(t_sust(end));  % Linear Mass change over burnout time
mass_sustboost = -t_sust.*dmdt+mint;
mass = mass_sustboost;

dt = t_sust(2)-t_sust(1);     % Time increment

for k = 1:length(t)-1
    rho(k+1) = Density(h(k));                    % Atmoshperic Density
    D(k+1) = Drag(v(k),h(k),A_sust,Cd);         % Function for drag (N)
    a(k+1) = acceleration(T_sust(k),D(k),mass_sustboost(k)); % Acceleration function to
find dvdt
    v(k+1) = v(k) + a(k)*dt;                     % Euler's Method for ODE's to get
velocity
    h(k+1) = h(k) + v(k)*dt;                     % Euler's Method for ODE's to get height
end
```

Sustainer Coast phase

```
apogee = 50;
t_coast2 = linspace(t(end),apogee,fidelity); % Coast Time
dt = t_coast2(2)-t_coast2(1);
t_coast2 = t_coast2+dt;                      % Coast Time

for k = length(t)-1:length(t)+length(t_coast2)-1
    mass(k+1) = mfin;
    rho(k+1) = Density(h(k));                    % Atmoshperic Density
    D(k+1) = Drag(v(k),h(k),A_sust,Cd);          % Function for drag (N)
    a(k+1) = acceleration(0,D(k),mfin);          % Acceleration function to find dvdt
    v(k+1) = v(k) + a(k)*dt;                     % Euler's Method for ODE's to get
velocity
    h(k+1) = h(k) + v(k)*dt;                     % Euler's Method for ODE's to get height
end

t = horzcat(t,t_coast2);
```

Dynamic Pressure

```
for k = 1:length(v)
    q(k) = .5*rho(k)*v(k)^2;        % Aerodynamic Pressure (Pa)
end
```

Displaying Data

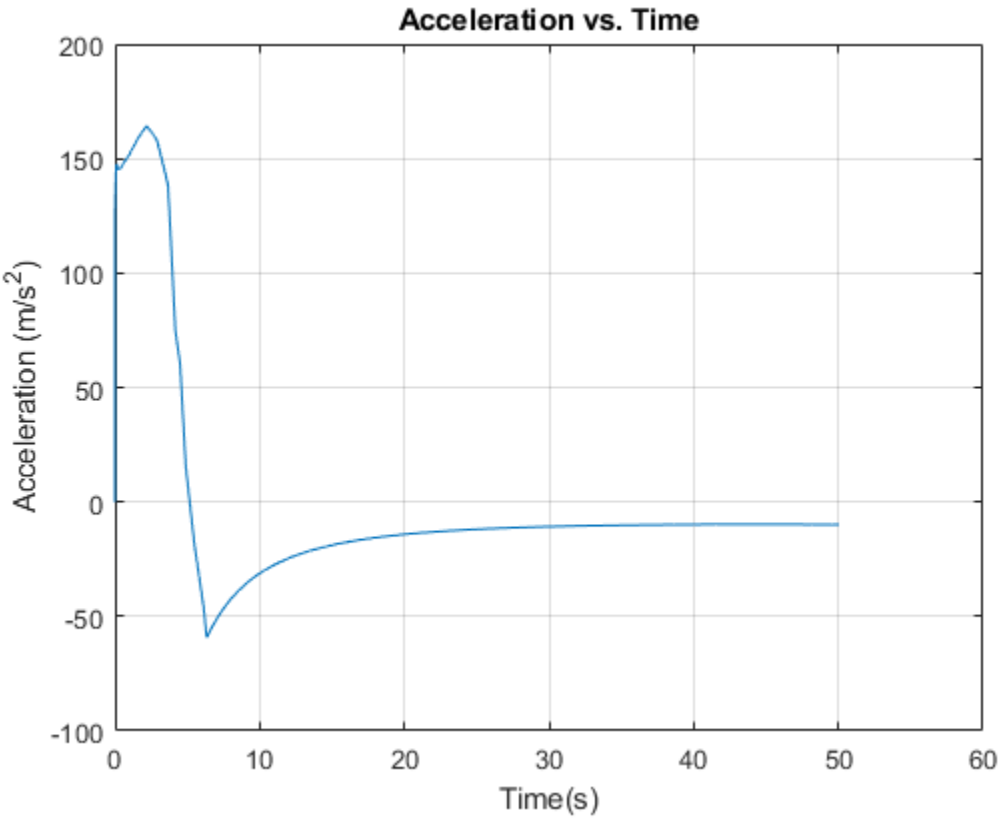
```
[z,y] = max(h);
fprintf('Maximum height is %0.f (m) or %0.f (ft) at %.0f (s)\n',z,z*3.281,t(y))
[z,y] = max(v);
fprintf('Maximum velocity is %0.f (m/s) or %0.f (ft/s) at %.0f (s)\n',z,z*3.281,t(y))
[z,y] = max(a);
fprintf('Maximum acceleration is %0.f (m/s^2) or %0.f (ft/s^2) at %.0f
(s)\n',z,z*3.281,t(y))
[maxq,index_maxq] = max(q);
fprintf('Maximum Dynamic Pressure: %0.f (Pa) at %.0f (s) and %0.f (m-
AGL)\n',maxq,t(index_maxq),h(index_maxq))
```

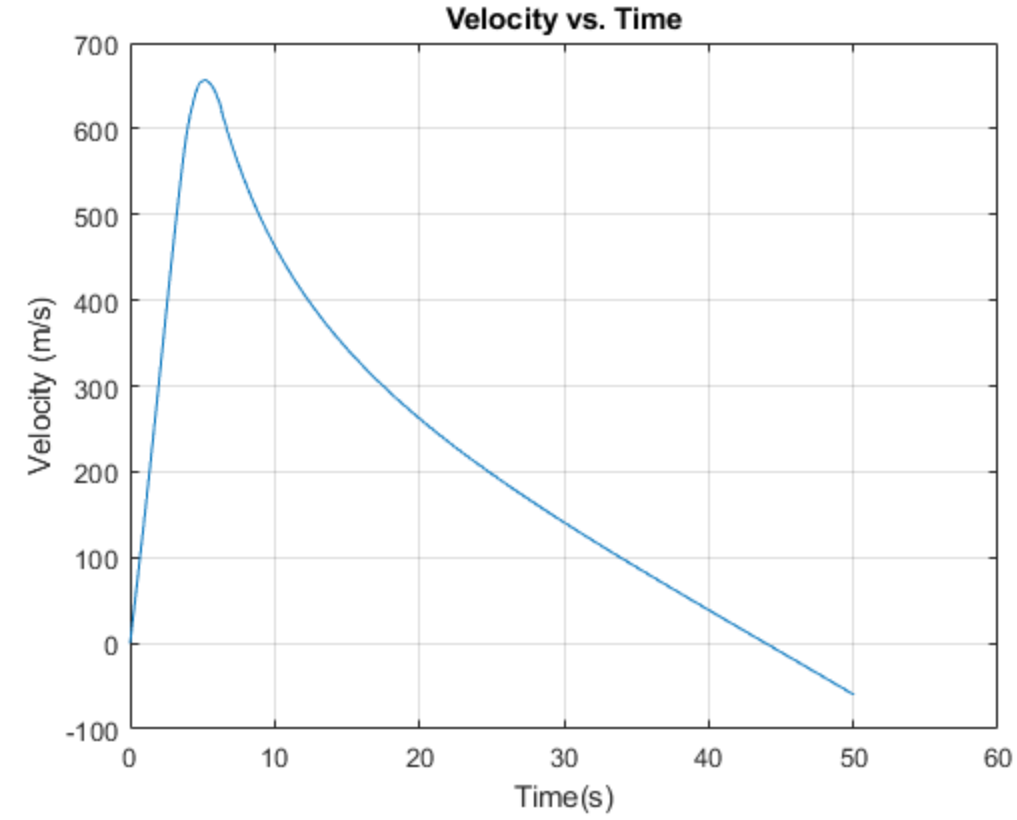
```
Maximum height is 11785 (m) or 38666 (ft) at 44 (s)
Maximum velocity is 657 (m/s) or 2154 (ft/s) at 5 (s)
Maximum acceleration is 164 (m/s^2) or 538 (ft/s^2) at 2 (s)
Maximum Dynamic Pressure: 193067 (Pa) at 5 (s) and 2342 (m-AGL)
```

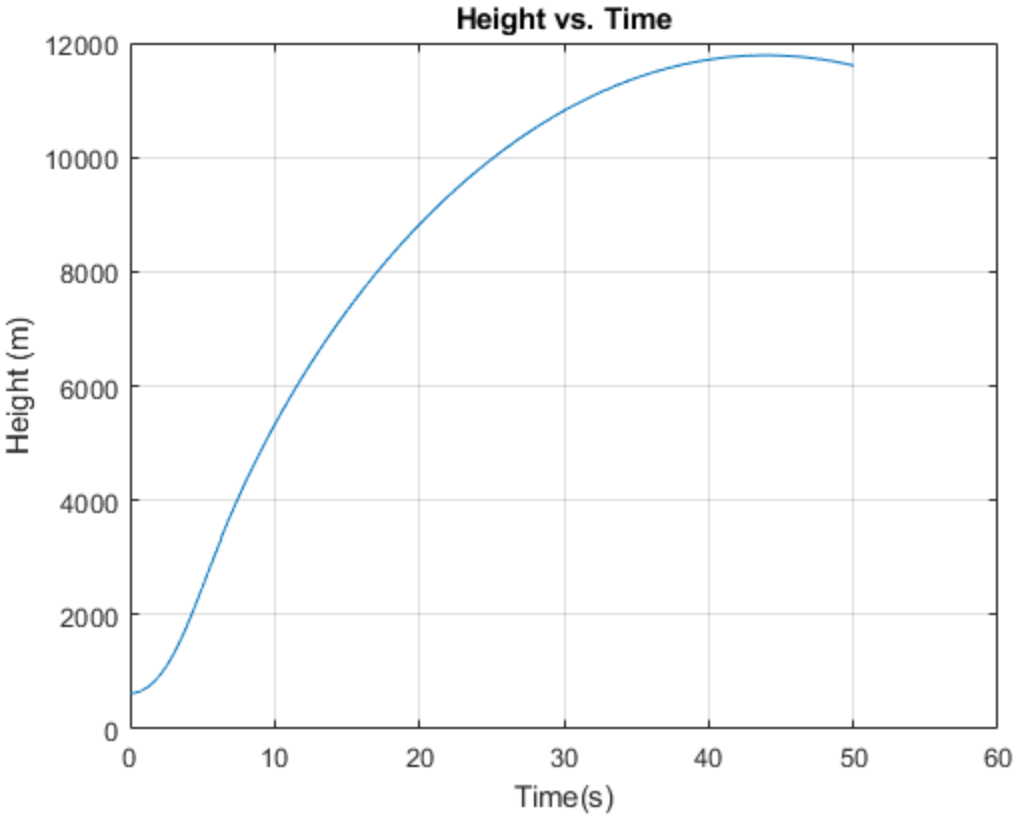
Plotting the Data

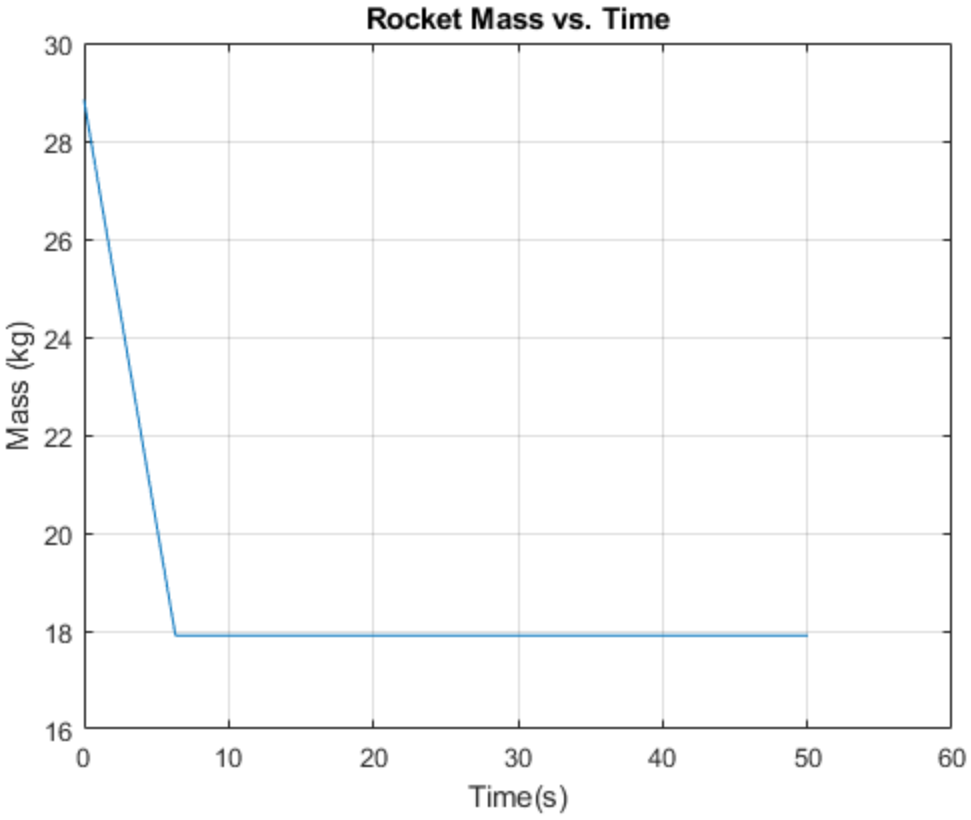
```
figure(1)
plot(t,a), xlabel('Time(s)'), ylabel('Acceleration (m/s^2)'), title('Acceleration vs.
Time')
grid on
figure(2)
plot(t,v), xlabel('Time(s)'), ylabel('Velocity (m/s)'), title('Velocity vs. Time')
grid on
```

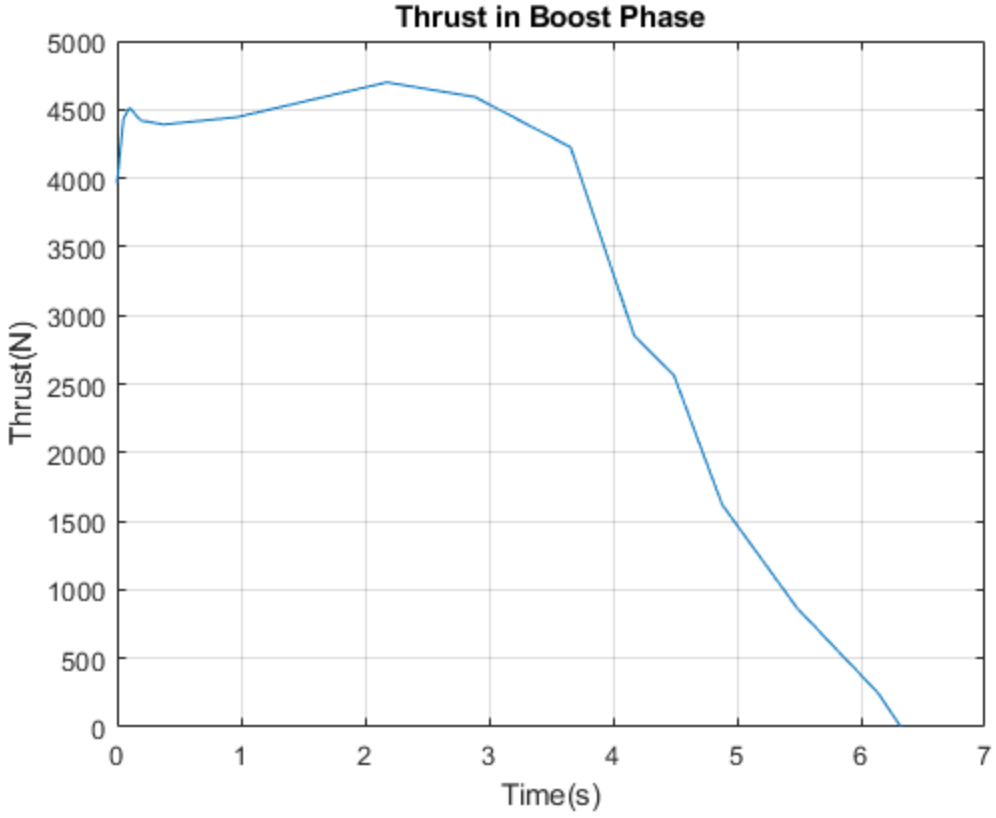
```
figure(3)
plot(t,h), xlabel('Time(s)'), ylabel('Height (m)'), title('Height vs. Time')
grid on
figure(4)
plot(t,mass), xlabel('Time(s)'), ylabel('Mass (kg)'), title('Rocket Mass vs. Time')
grid on
figure(5)
plot(t_sust,T_sust), xlabel('Time(s)'), ylabel('Thrust(N)'), title('Thrust in Boost
Phase')
grid on
figure(6)
plot(t,rho), xlabel('Time(s)'), ylabel('Air Density (kg/m^3)'), title('Air Density vs.
Time')
grid on
figure(7)
subplot(2,1,1)
plot(t,q,t(index_maxq),maxq,'or'), xlabel('Time(s)'), ylabel('Aerodynamic Pressure
(Pa)'), title('Dynamic Pressure vs. Time')
legend('Dynamic Pressure (Pa)','Max Q')
grid on
subplot(2,1,2)
plot(h,q,h(index_maxq),maxq,'or'), xlabel('Altitude (m)'), ylabel('Aerodynamic Pressure
(Pa)'), title('Dynamic Pressure vs. Altitude')
legend('Dynamic Pressure (Pa)','Max Q')
grid on
```

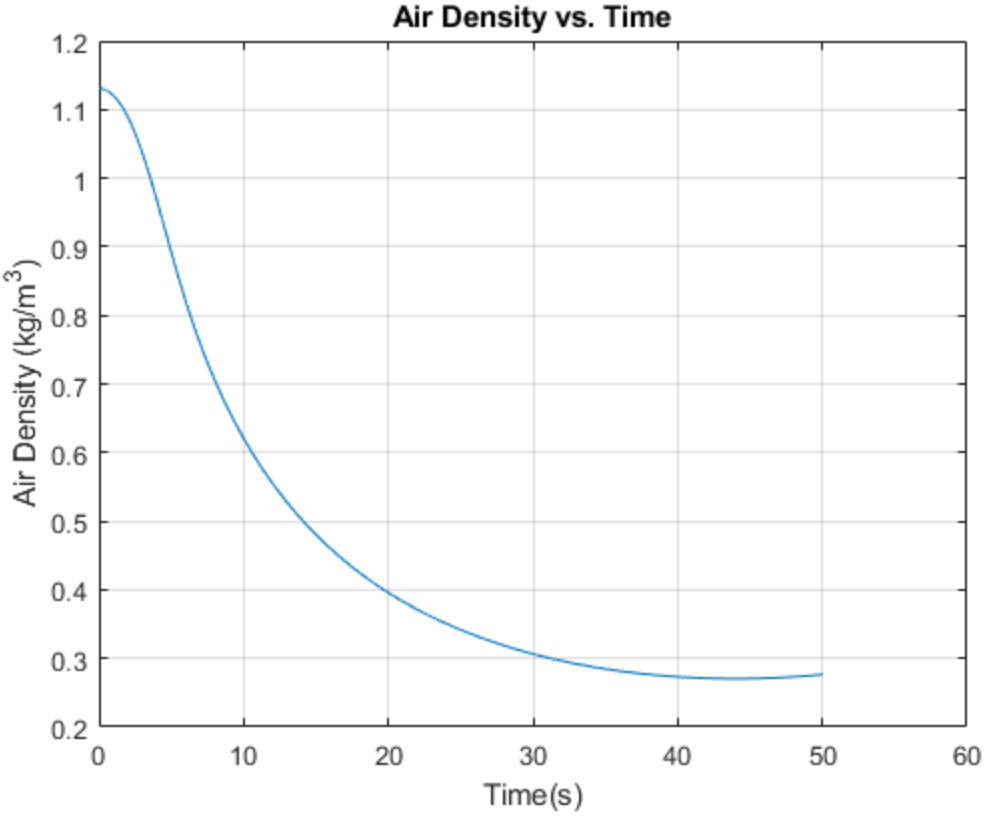


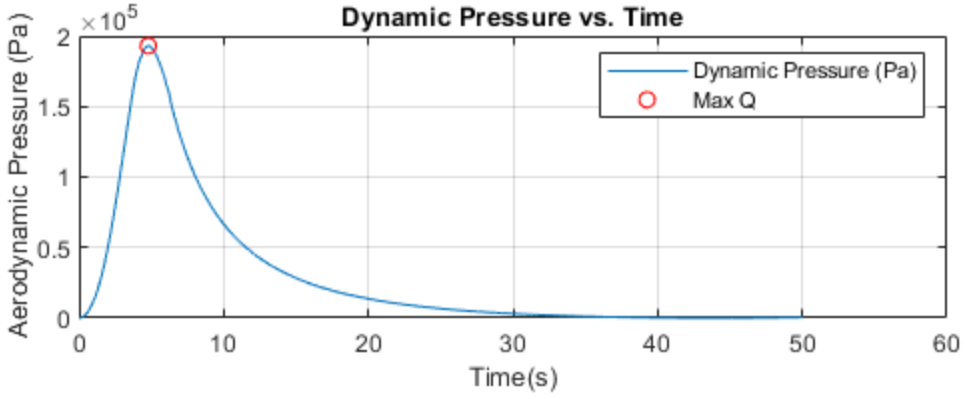

Velocity vs. Time
Velocity (m/s)
Time(s)
700
600
500
400
300
200
100
0
-100
0
10
20
30
40
50
60

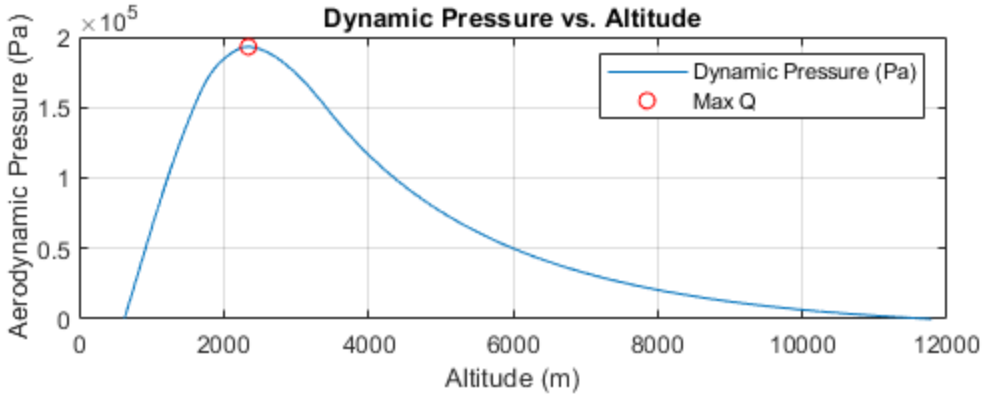

Height vs. Time
Height (m)
Time(s)
12000
10000
8000
6000
4000
2000
0
0
10
20
30
40
50
60


Rocket Mass vs. Time
Mass (kg)
Time(s)
30
28
26
24
22
20
18
16
0
10
20
30
40
50
60


Thrust in Boost Phase
Thrust(N)
Time(s)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
7


Air Density vs. Time
Air Density (kg/m^3)
Time(s)
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
60


Dynamic Pressure vs. Time
×10^5
Aerodynamic Pressure (Pa)
Time(s)
Dynamic Pressure (Pa)
Max Q
2
1.5
1
0.5
0
0
10
20
30
40
50
60


Dynamic Pressure vs. Altitude
×10^5
Aerodynamic Pressure (Pa)
Altitude (m)
Dynamic Pressure (Pa)
Max Q
2
1.5
1
0.5
0
0
2000
4000
6000
8000
10000
12000

Functions

```
function [ D ] = Drag( v,h,A,Cd )
%The Drag function calculates the drag on an object at a certain velocity

D = .5*Density(h)*v^2*Cd*A;   % Function for drag (N)


end

function [rho] = Density(h)
% Function for changing atmospheric density
global rhos;

rho = rhos*exp((-1/7800)*h);

end

function [ dvdt ] = acceleration(T,D,M)
% The acceleration function is used to calculate the acceleration of a
% rocket at a given time and velocity.

global g0;

dvdt = (T - D)/M - g0;



end
```

C. MODIFIED BARROWMAN STABILITY CALCULATIONS

Modified Barrowman Stability Calculations

The program Barrowman_Stability is used to estimate the location of the rocket's center of pressure (CP) using the Barrowman Stability Equations. In this example, the script is used to calculate the CP for the NPS SSAG high-power rocket (two-stage configuration) using the initial sustainer (flight 5) and booster designs. The single-stage configuration can be calculated by setting the calc_twostage parameter to false. Additionally, the CP for a Von Karman profile is calculated in this script as Barrowman does not provide an estimate for the nose cone term for this specific shape of nose cone in his analysis.

Source Author: Dillon Pierce
Name of File: Barrowman_Stability.m
Location of File: ssagcommon$\High Power Rocket\MATLAB\
Date Last Modified: 23 May 2019

Inputs: Various physical parameters for the rocket
Output: Barrowman estimate for CP in inches from tip of nose cone

Program Start

```
clear, clc, format compact
```

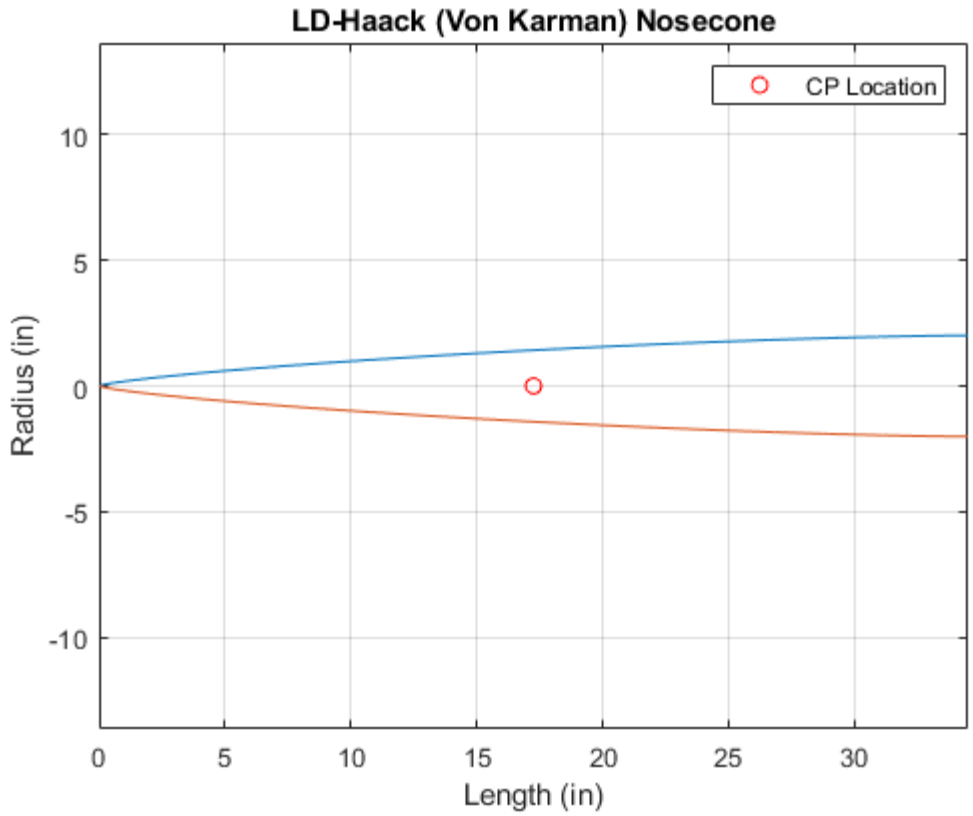
Rocket Inputs

```
calc_twostage = true;   % Calculate sustainer alone or two stage configuration

% Sustainer Airframe and Fin Inputs

Ln = 34.5;              % Length of nose cone (in)
d = 6.17;               % Diameter at base of nose (in)
df = 6.17;              % Diameter at front of transition (in)
dr = 4.024;             % Diameter at rear of transition (in)
Lt = 4;                 % Length of transition (in)
Xp = 34.5;              % Distance from tip of nose to front of transition (in)
Cr = 10;                % Fin root chord (in)
Ct = 2;                 % Fin tip chord (in)
S = 4.625;              % Fin semispan (in)
Lf = 5.58;              % Length of fin mid-chord line (in)
R = 2.01;               % Radius of body at aft end (in)
Xr = 7.125;             % Distance between fin root leading edge and fin tip leading edge
parallel to body (in)
Xb = 109.5;             % Distance from nose tip to fin root chord leading edge (in)
N = 3;                  % Number of fins

% Booster Airframe and Fin Inputs

b_d = 6.17;               % Diameter at base of nose (in)
b_df = 4.024;             % Diameter at front of transition (in)
```

```
b_dr = 6;                   % Diameter at rear of transition (in)
b_Lt = 3;                   % Length of transition (in)
b_Xp = 119.5;               % Distance from tip of nose to front of transition (in)
b_Cr = 13.5;                % Fin root chord (in)
b_Ct = 2.6;                 % Fin tip chord (in)
b_S = 6.5;                  % Fin semispan (in)
b_Lf = 7.73;                % Length of fin mid-chord line (in)
b_R = 3;                    % Radius of body at aft end (in)
b_Xr = 9.625;               % Distance between fin root leading edge and fin tip leading
edge parallel to body (in)
b_Xb = 245.6;               % Distance from nose tip to fin root chord leading edge (in)
b_N = 3;                    % Number of fins

% Nose Cone Parameter Inputs

Rad = 6.17/2;           % Radius of nose cone at base (in)
L = 34.5;               % Length of nose cone (in)
fidelity = 1000;        % Fidelity of calculation
```

Von Karman LD-Haack Nose Cone Term Calculations

```
theta = zeros(1,fidelity);
y = zeros(1,fidelity);
x = linspace(0,L,fidelity);
neg_y = zeros(1,fidelity);
vol_y = zeros(1,fidelity);
dx = L/fidelity;
y(1) = 0;

for k = 1:fidelity
    theta(k) = acos(1-(2*x(k))/L);
    y(k) = (R/sqrt(pi))*sqrt(theta(k)-(sin(2*theta(k)))/2);
    neg_y(k) = -(R/sqrt(pi))*sqrt(theta(k)-(sin(2*theta(k)))/2);
    vol_y(k+1) = vol_y(k)+pi*y(k)^2*dx;

end

volume = vol_y(end);

Z = (L-(volume/(pi*y(end)^2)));

figure(1)
plot(x(k)*(Z/L),0,'or',x,y,x,neg_y)
title('LD-Haack (Von Karman) Nose cone'), xlabel('Length (in)'), ylabel('Radius (in)')
axis('equal')
grid on
legend('CP Location')
```



Nose Cone Term

```
Xn = (Z/L)*Ln;          % For Von Karman Nose cone
Cnn = 2;
```

Conical Transition Terms

Sustainer Terms

```
Cnt = 2*((dr/d)^2-(df/d)^2);
Xt = Xp+(Lt/3)*(1+((1-(df/dr))/(1-(df/dr)^2)));

% Booster Terms
b_Cnt = 2*((b_dr/b_d)^2-(b_df/b_d)^2);
b_Xt = b_Xp+(b_Lt/3)*(1+((1-(b_df/b_dr))/(1-(b_df/b_dr)^2)));
```

Fin Terms

```
Cnf = (1+(R/(S+R)))*((4*N*(S/d)^2)/(1+(sqrt(1+(((2*Lf)/(Cr+Ct))^2)))));
Xf = Xb+(((Xr/3)*(Cr+2*Ct))/(Cr+Ct))+((1/6)*((Cr+Ct)-((Cr*Ct)/(Cr+Ct))));

b_Cnf =
(1+(b_R/(b_S+b_R)))*((4*b_N*(b_S/b_d)^2)/(1+(sqrt(1+(((2*b_Lf)/(b_Cr+b_Ct))^2)))));
b_Xf = b_Xb+(((b_Xr/3)*(b_Cr+2*b_Ct))/(b_Cr+b_Ct))+((1/6)*((b_Cr+b_Ct)-
((b_Cr*b_Ct)/(b_Cr+b_Ct))));
```

Center of Pressure

```
if calc_twostage == true
    Cnr = Cnn + Cnf + Cnt + b_Cnf + b_Cnt;
    X = ((Cnn*Xn)+(Cnf*Xf)+(Cnt*Xt)+(b_Cnf*b_Xf)+(b_Cnt*b_Xt))/Cnr;
else
    Cnr = Cnn + Cnf + Cnt;
    X = ((Cnn*Xn)+(Cnf*Xf)+(Cnt*Xt))/Cnr;               % Location from Nose Tip
end


fprintf('LD-Haack Barrowman Parameter = %.3f \n',Z/L)
fprintf('Center of Pressure Distance from Nose Cone = %3.2f in\n', X)
```

```
LD-Haack Barrowman Parameter = 0.500
Center of Pressure Distance from Nose Cone = 184.58 in
```

D. PARACHUTE DESCENT RATE CALCULATOR

Parachute Descent Rate Calculator

The Parachute script is used to calculate the descent rate of the various rocket components using a dual-deployment technique. The example output for this specific script shows the estimated descent rate parameters for the redesigned sustainer at an apogee altitude of 260,000 ft.

Source Author: Dillon Pierce
Name of File: Parachute.m
File Location: ssagcommon$\High Power Rocket\MATLAB
Date Last Modified: 23 May 2019

Inputs: Drogue and main parachute physical dimensions and coefficients of drag, section mass, and apogee altitude.
Outputs: Descent rate at main parachute opening and descent rate at landing

Program Start

Program start will clean the command window, workspace variables, and format MATLAB output in a compact mode

```
clear, clc, format compact
```

Given Parachute Characteristics

```
d = 24;                 % (Input) Drogue Parachute diam (in)
Cd = 1.55;              % (Input) Coefficient of Drag (Drogue)
mass = 25;              % (Input) Rocket Empty Mass (lbs)
Apogee = 260000;        % (Input) Max Altitude (ft)
main_Cd = 2;            % (Input) Main Parachute Cd
main_d = 96;            % (Input) Main Parachute diameter (in)
```

Constants Assumed

```
rhos = 1.225;       % Density of air in kg/m^3 at 20 deg. Centigrade
g0 = 9.807;         % Gravity constant (m/s^2)
Re = 6378;          % Radius of Earth (km)
```

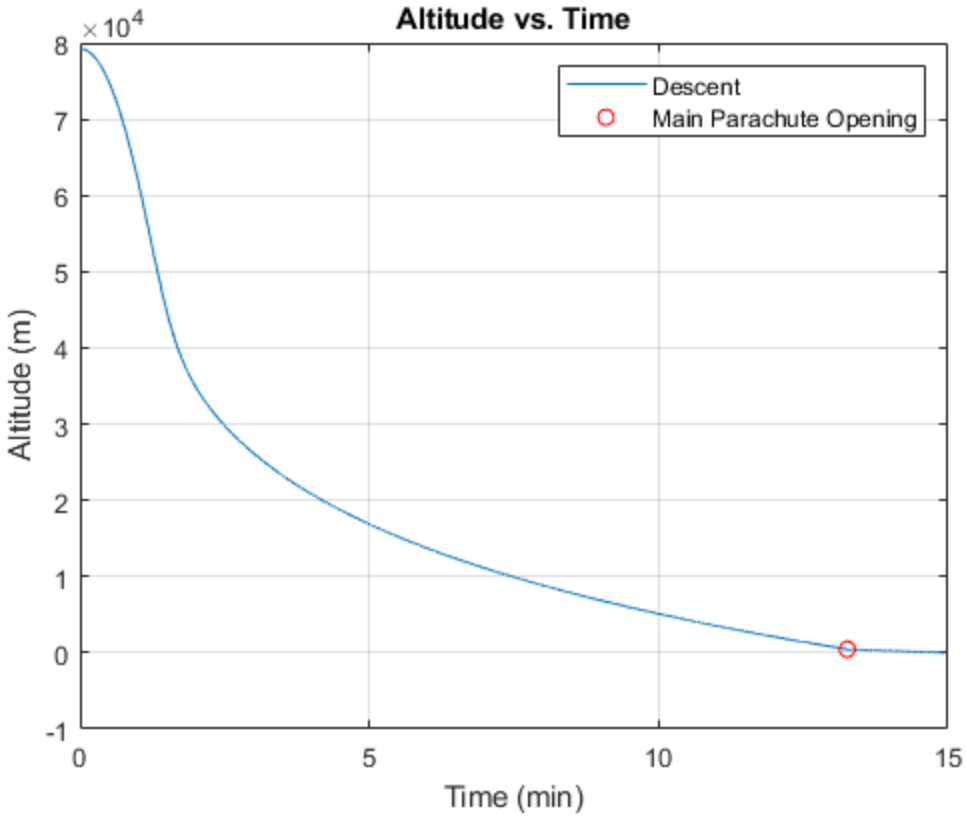
Conversion Calculations

```
d = d*0.0254;               % Parachute diam (m)
mass = mass/2.2;            % Rocket Mass (kg)
h(1) = 0.3048*Apogee;       % Max Altitude (m)
A = pi*(d*.5)^2;            % Cross-sectional Area of paracute (m^2)
d_main = main_d*0.0254;     % Main Parachute diam(m)
A_main = pi*(d_main*.5)^2;  % Cross-sectional Area of main paracute (m^2)
```

Descent Calculations

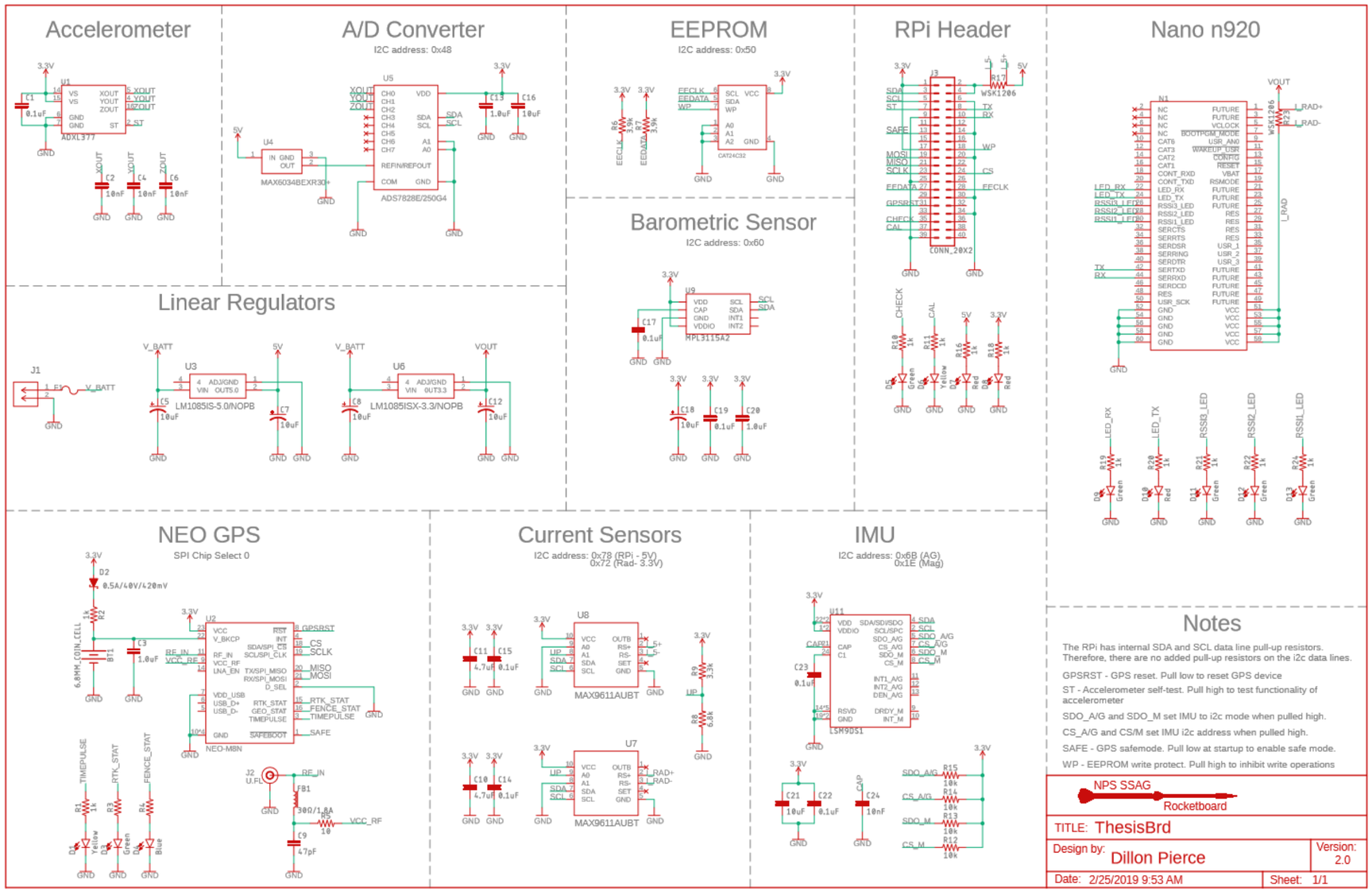
```
rho(1) = rhos;
v(1) = 0;
gh(1) = g0;
dt = .1;
check = 0;

for k = 1:100000
    rho(k) = rhos*exp((-1/8000)*h(k));
    D(k) = .5*rho(k)*v(k)^2*Cd*A;
    dvdt(k) = (D(k)/mass)-g0;
    Vterm(k) = sqrt((2*mass*g0)/(rho(k)*A*Cd));
    if norm(v(k)) >= norm(Vterm)
        v(k) = Vterm(k);
        v(k+1) = Vterm(k);
        h(k+1) = h(k) + v(k)*dt;
    else
        v(k+1) = v(k) + dvdt(k)*dt;
        h(k+1) = h(k) + v(k)*dt;
    end
    if h(k) <= 457
        Cd = main_Cd;
        A = A_main;
        if check == 0
        fprintf('Descent Rate at Main Open = %2.2f m/s or %2.2f fps\n',v(k),v(k)*3.28084)
        index = k;
        check = 1;
        end
    end
    if h(k) <= 0
        fprintf('Descent Rate at Landing = %2.2f m/s or %2.2f fps\n',v(k),v(k)*3.28084)
        break
    end
end

t = 1:length(h);
fprintf('Total Elapsed Time = %.0f sec or %.2f min \n',t(end)*(1/dt),t(end)/600)
plot((t/(60/dt)),h,t(index)/(60/dt),h(index),'or')
title('Altitude vs. Time'), xlabel('Time (min)'), ylabel('Altitude (m)')
legend('Descent','Main Parachute Opening')
grid on
```

```
Descent Rate at Main Open = -20.66 m/s or -67.79 fps
Descent Rate at Landing = -4.41 m/s or -14.48 fps
Total Elapsed Time = 89690 sec or 14.95 min
```


Altitude vs. Time
×10^4
8
7
6
5
4
3
2
1
0
-1
Altitude (m)
0
5
10
15
Time (min)
Descent
Main Parachute Opening

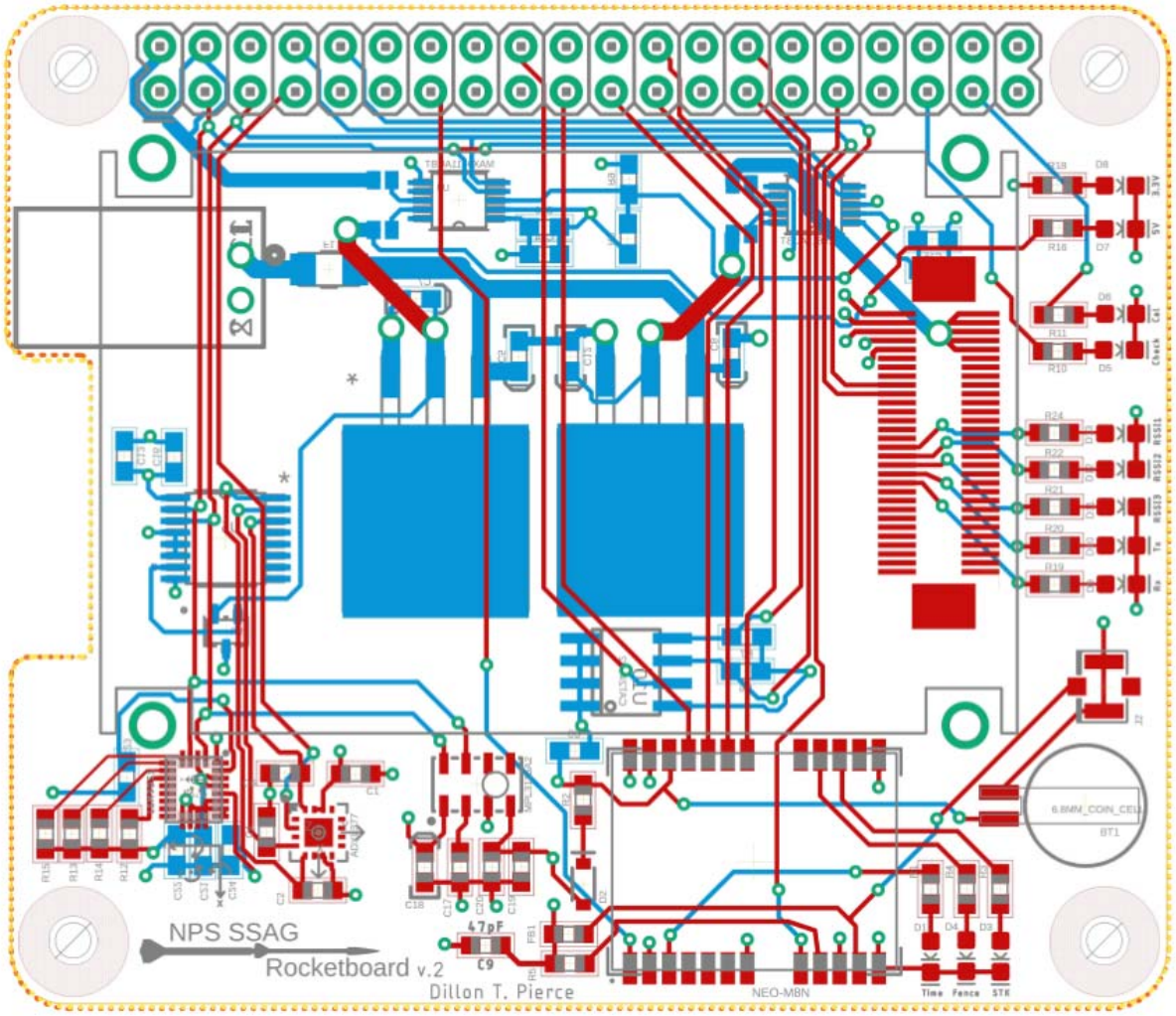
THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. ROCKET SENSOR BOARD SCHEMATIC AND PCB[6]

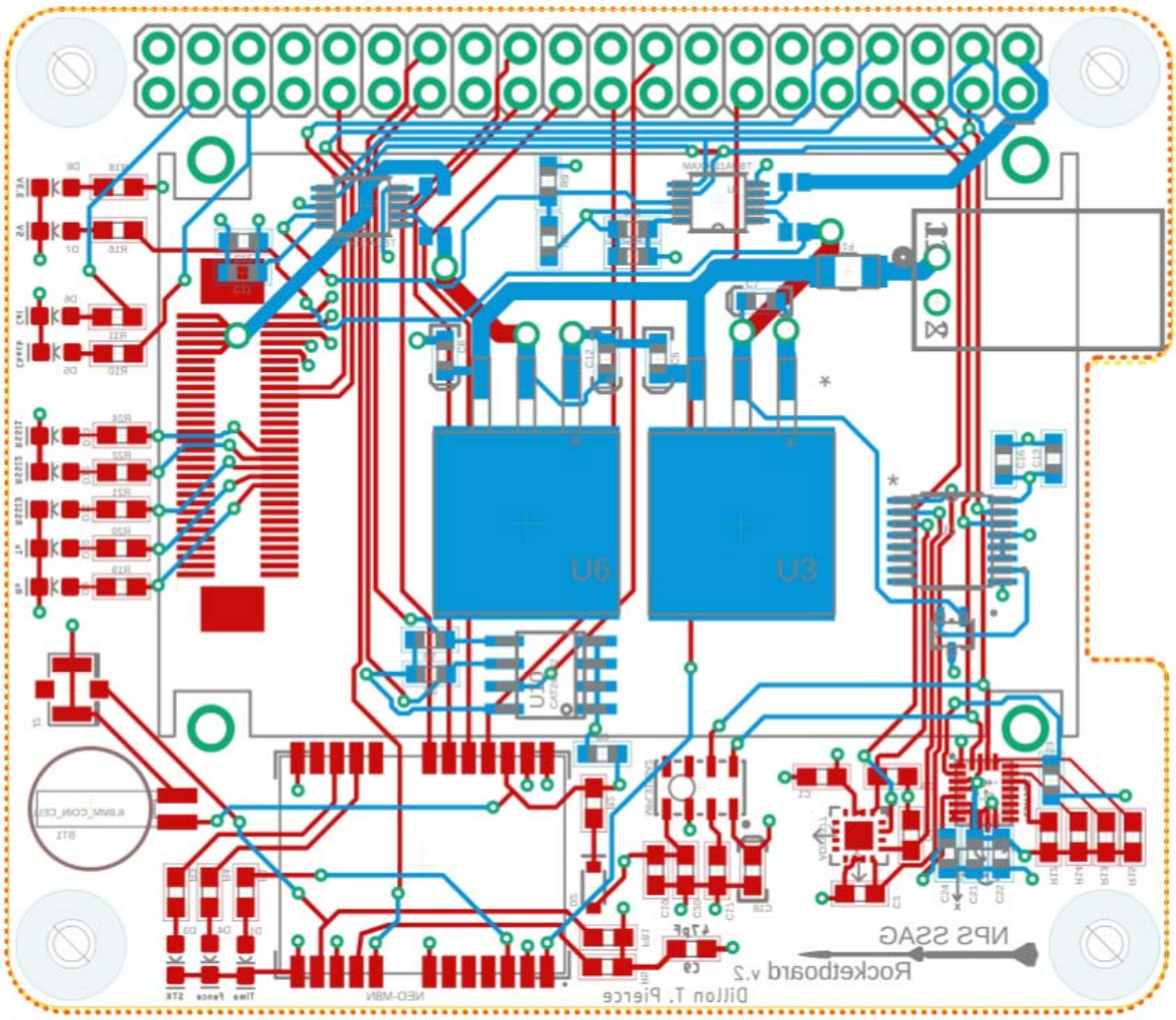


[6] Schematic and board files available: https://github.com/dpierce1274/NPS-RocketBoard.git

A. PCB TOP VIEW



B. PCB BOTTOM VIEW



THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SENSOR BOARD SOFTWARE

A. MAIN.PY

The Main flight software is the primary software script for the NPS SSAG Rocketboard. The software is manually initiated (and back grounded!) via a WiFi SSH terminal. The script captures the rocket's various flight sensor data and stores the data in a .txt file. The script also initiates a sub process (Telemetry.py) that is responsible for capturing and sending GPS telemetry data via the MHX radio.

Source Author: Dillon Pierce
Name of File: Main.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

```
import time
import sys
import mpl3115a2
import adxl377
import traceback
import IMU
from gpiozero import LED
import subprocess
import shlex

g_counter = 0


def main():

    # Declare global variables
    global g_counter
    launch_indicator = False

    # Configure Status LEDs
    cal_led = LED(26)
    check_led = LED(19)
    cal_led.on()

    # I. Program initialization
    # Define initialization variables
    baro = mpl3115a2.MPL3115A2(busID=1, slaveAddr=0x60,
sea_level_pressure=1012.0)
    acc = adxl377.ADXL377(busID=1, slaveAddr=0x48)

    # Initialize the sensors
    IMU.initIMU()

    # Begin Telemetry Process
```

```
    p_name = '/usr/bin/python Telemetry.py'
    process = subprocess.Popen(shlex.split(p_name),
stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    # Create data file and write header #
    tstr = time.strftime('%Y-%m-%d_%H-%M-%S-%Z.txt')
    filename = str('Flight Data ') + tstr
    header_1 = str('Counter, Time, ACCx, ACCy, ACCz, accx, accy, accz,
GRYx, GRYy, GRYz, MAGx, MAGy, MAGz, '
                   'Temp(C), Pres(mbar), Alt (m) \n')
    fp = open(filename, 'a')
    fp.write(header_1)
    fp.close()

    time.sleep(1)
    cal_led.off()
    # II. Main Loop

    while True:
        g_counter += 1                                  # Iterate
Counter
        status_led(check_led)                           # Flash status
LED
        flt_params = read_flt_params(baro, IMU, acc)    # Get the
flight parameters

        write_to_file(filename, flt_params)             # Append
parameters to file


def read_flt_params(baro, IMU, acc):
    global g_counter

    baro_data = baro.get_barometer_data()
    imu_data = IMU.get_IMU_data()
    acc_data = acc.get_accel_values()
    ACCx = float(imu_data[0])
    ACCy = float(imu_data[1])
    ACCz = float(imu_data[2])
    accx = float(acc_data[0])
    accy = float(acc_data[1])
    accz = float(acc_data[2])
    GRYx = float(imu_data[3])
    GRYy = float(imu_data[4])
    GRYz = float(imu_data[5])
    MAGx = float(imu_data[6])
    MAGy = float(imu_data[7])
    MAGz = float(imu_data[8])
    temp = float(baro_data[0])
    pres = float(baro_data[1])
    alt = int(baro_data[2])

    t = float('{:.2f}'.format(time.time()))
    output = [g_counter, t, ACCx, ACCy, ACCz, accx, accy, accz, GRYx,
GRYy, GRYz, MAGx, MAGy, MAGz, temp, pres, alt]
```

```
    return output


def write_to_file(filename, flt_params):
    # This function writes the flight parameters array to a .txt file
on the SD Card
    # Input: flight parameters
    # Output: none
    fp = open(filename, 'a+')
    fp.write(str(flt_params) + '\n')
    fp.close()


def status_led(check_led):
    global g_counter

    if g_counter % 200 == 0:
        check_led.on()
        print('Main enabled')
    elif g_counter % 100 == 0:
        check_led.off()


start = time.time()

# Execute `main()` function
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print("Program Terminating...")
        time.sleep(1)
        sys.exit()
    except:                                                    # Pass
and log all other errors
        s = traceback.format_exc()
        fp = open("Traceback_Log.txt", "a")
        fp.write(s + "\n")
        fp.close()
```

B. TELEMETRY.PY

Telemetry.py is a sub process that runs under Main.py that is responsible for capturing, sending, and storing GPS data. Telemetry data is sent via serial communications with the MHX radio to the ground station. Currently, the software is only configured for sending data. Adding a serial read capability to this script would enable ground station up link to the rocket.

Source Author: Dillon Pierce
Name of File: Telemetry.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

```
import traceback
import serial
import ublox
import time
import sys


def main():

    ser = serial.Serial(port='/dev/ttyS0', baudrate=9600, bytesize=8,
parity='N', stopbits=1, timeout=0.01)

    # Create data file and write header #
    tstr = time.strftime('%Y-%m-%d_%H-%M-%S-%Z.txt')
    filename = str('Telemetry Data ') + tstr

    ubl = ublox.UBlox("spi:0.0", baudrate=5000000, timeout=2)

    ubl.set_preferred_dynamic_model(None)
    ubl.set_preferred_usePPP(None)

    # Configure the GPS messages
    ubl.configure_solution_rate(rate_ms=1000)
    ubl.configure_message_rate(ublox.CLASS_NAV, ublox.MSG_NAV_POSLLH,
1)
    ubl.configure_message_rate(ublox.CLASS_NAV, ublox.MSG_NAV_STATUS,
1)
    ubl.configure_message_rate(ublox.CLASS_NAV, ublox.MSG_NAV_VELNED,
1)
    ubl.configure_message_rate(ublox.CLASS_NAV, ublox.MSG_NAV_PVT, 1)

    message_names = ['NAV_POSLLH', 'NAV_STATUS', 'NAV_VELNED',
'NAV_PVT']

    check_msg = []

    while True:
        msg = ubl.receive_message()
        print(msg)
        if msg is None:
            if opts.reopen:
                ubl.close()
```

```
                ubl = ublox.UBlox("spi:0.0", baudrate=5000000,
timeout=2)
                continue
            print(empty)
            break
        if msg.name() == "NAV_STATUS":
            outstr = str(msg).split(",")
            fix_id = int(str_to_num(outstr[1]))  # GPS fix - from UBX-
NAV-STATUS function
            fix_ok = int(str_to_num(outstr[3]))  # gpsFixOk (1 =
position and velocity valid and within DOP and ACC)
            write_to_file(filename, outstr)
            check_msg.append(msg.name())
            print(outstr)
        if msg.name() == "NAV_POSLLH":
            outstr = str(msg).split(",")
            lon = float('%.7f' % (str_to_num(outstr[1])*10**-7))
# GPS longitude (deg)
            lat = float('%.7f' % (str_to_num(outstr[2])*10**-7))
# GPS latitude (deg)
            h_msl = float('%.1f' % (int(str_to_num(outstr[4])/1000)))
# GPS height MSL (m)
            write_to_file(filename, outstr)
            check_msg.append(msg.name())
        if msg.name() == "NAV_VELNED":
            outstr = str(msg).split(",")
            gps_hdg = float('%.1f' % (str_to_num(outstr[6])*10**-5))
# GPS heading (deg)
            gps_gspd = int(str_to_num(outstr[5])/100)
# GPS ground speed (m/s)
            write_to_file(filename, outstr)
            check_msg.append(msg.name())
        if msg.name() == "NAV_PVT":
            outstr = str(msg).split(",")
            hour = outstr[4]
            min = outstr[5]
            sec = outstr[6]
            gps_time = '{}{}{}'.format(hour,min,sec)
            write_to_file(filename, outstr)
            check_msg.append(msg.name())

        if all(elem in check_msg for elem in message_names):
            data = [gps_time, 'NAVMSG', int(time.time()), lat, lon,
h_msl]
            write_to_file(filename, data)
            send_gps(ser, data)
            check_msg = []


def send_gps(ser, data):
    # This function formats and sends a GPS telemetry message to the
ground radio
    # Inputs: GPS
    # Outputs: none
```

```
    msg = "{:>10s} {:>6s} {:>10d} {:>+9.5f} {:>+10.5f} deg {:>5d}
m".format(data[0], data[1], int(data[2]), data[3],

data[4], int(data[5]))
    ser.write('{:<100s}'.format('12 %s\r' % msg))
    print('{:<100s}'.format('12 %s\n' % msg))


def str_to_num(input):
    # This function converts a GPS string out value with equals sign to
a float value
    # Inputs: String value with =
    # Outputs: Float value

    # Find the index of the equals sign
    index = input.find('=')

    # Remove all characters up to and including the equals sign
    output = input[index + 1:]

    output = float(output)

    return output


def write_to_file(filename, flt_params):
    # This function writes the flight parameters array to a .txt file
on the SD Card
    # Input: flight parameters
    # Output: none
    fp = open(filename, 'a')
    fp.write(str(flt_params) + '\n')
    fp.close()


start = time.time()

# Execute `main()` function
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print("Program Terminating...")
        time.sleep(1)
        sys.exit()
    except:                                                    # Pass
and log all other errors
        s = traceback.format_exc()
        fp = open("GPS_Traceback_Log.txt", "a")
       fp.write(s + "\n"
       fp.close()
```

C. MPL3115A2.PY

Mpl3115a2 is the Python object for the mpl3115a2 barometric sensor. The sensor communicates with the RPi via i2c. Altitude is computed using an equation provided in the sensor's documentation.

Source Author: Dillon Pierce
Name of File: mpl3115a2.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

Inputs: BusID, Slave Address, Sea level pressure (mbar)
Outputs: temperature (deg C), pressure (mbar), altitude (m)

```
import smbus
import time

# Register Address Map
STATUS           = 0x00
OUT_P_MSB        = 0x01
OUT_P_CSB        = 0x02
OUT_P_LSB        = 0x03
OUT_T_MSB        = 0x04
OUT_T_LSB        = 0x05
DR_STATUS        = 0x06
OUT_P_DELTA_MSB  = 0x07
OUT_P_DELTA_CSB  = 0x08
OUT_P_DELTA_LSB  = 0x09
OUT_T_DELTA_MSB  = 0x0A
OUT_T_DELTA_LSB  = 0x0B
WHO_AM_I         = 0x0C
F_STATUS         = 0x0D
F_DATA           = 0x0E
F_SETUP          = 0x0F
TIME_DLY         = 0x10
SYSMOD           = 0x11
INT_SOURCE       = 0x12
PT_DATA_CFG      = 0x13
BAR_IN_MSB       = 0x14
BAR_IN_LSB       = 0x15
P_TGT_MSB        = 0x16
P_TGT_LSB        = 0x17
T_TGT            = 0x18
P_WND_MSB        = 0x19
P_WND_LSB        = 0x1A
T_WND            = 0x1B
P_MIN_MSB        = 0x1C
P_MIN_CSB        = 0x1D
P_MIN_LSB        = 0x1E
T_MIN_MSB        = 0x1F
T_MIN_LSB        = 0x20
P_MAX_MSB        = 0x21
```

```
P_MAX_CSB       = 0x22
P_MAX_LSB       = 0x23
T_MAX_MSB       = 0x24
T_MAX_LSB       = 0x25

# Control Registers
CTRL_REG1 = 0x26
CTRL_REG2 = 0x27
CTRL_REG3 = 0x28
CTRL_REG4 = 0x29
CTRL_REG5 = 0x2A

# Data Offsets
OFF_P = 0x2B
OFF_T = 0x2C
OFF_H = 0x2D



class MPL3115A2(object):

    def __init__(self, busID, slaveAddr, sea_level_pressure):
        self.__i2c = smbus.SMBus(busID)
# Set busid equal to user input ID
        self.__slave = slaveAddr
# Take given slave address
        self.__sea_level_pressure = sea_level_pressure
# Updated sea_level_pressure value (mbar)

        # No FIFO setup (Polling)
        self.__i2c.write_byte_data(self.__slave, CTRL_REG1, 0x38)
# Set Barometer standby mode with OSR 128
        self.__i2c.write_byte_data(self.__slave, PT_DATA_CFG, 0x07)
# Enable Data Flags
        self.__i2c.write_byte_data(self.__slave, CTRL_REG1, 0x39)
# Set Barometer active mode with OSR 128
        time.sleep(1)
# Allow sensor to enter active mode

    def get_barometer_data(self):
        # Read Registers
        out_p_msb = self.__i2c.read_byte_data(self.__slave, OUT_P_MSB)
        out_p_csb = self.__i2c.read_byte_data(self.__slave, OUT_P_CSB)
        out_p_lsb = self.__i2c.read_byte_data(self.__slave, OUT_P_LSB)
        out_t_msb = self.__i2c.read_byte_data(self.__slave, OUT_T_MSB)
        out_t_lsb = self.__i2c.read_byte_data(self.__slave, OUT_T_MSB)
        # Format Data
        pres = (out_p_msb << 16 | out_p_csb << 8 | out_p_lsb & 0xF0) /
64.0   # Convert pres data to 20-bit unsigned
        pres_mbar = pres / 100.0
# Convert pressure value to mbar
        temp = (out_t_msb << 8 | out_t_lsb & 0xF0) / 256.0      #
Convert temp data to 12-bit two's complement
        # Compute Altitude
        alt = 44330.77*(1-(pres_mbar/
```

```
                                self.__sea_level_pressure)**0.1902632)
# Compute altitude from documentation formula

        # Return Values
    return '{:.2f}'.format(temp), '{:.2f}'.format(pres_mbar),
'{:.0f}'.format(alt)
```

D. IMU.PY [80]

IMU is the Python object for the LSM9DS1 inertial measurement unit. The sensor communicates with the RPi via i2c.

Source Authors: Mark Williams and Peter Peck
Modified by: Dillon Pierce
Name of File: mpl3115a2.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

Inputs: None
Outputs: Acceleration-Gs (x,y,z), Rotation-deg/s (x,y,z)

```
import smbus
import time

bus = smbus.SMBus(1)
from LSM9DS1 import *


LSM9DS0 = 1
gain = 0.07         # Gyro Gain based on init setting

def detectIMU():
    try:
        # Check for LSM9DS1
        # If no LSM9DS1 is connected, there will be an I2C bus error
and the program will exit.
        # This section of code stops this from happening.
        LSM9DS1_WHO_XG_response =
(bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_WHO_AM_I_XG))
        LSM9DS1_WHO_M_response =
(bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_WHO_AM_I_M))

    except IOError as f:
        print('')  # need to do something here, so we just print a
space
    else:
        if (LSM9DS1_WHO_XG_response == 0x68) and
(LSM9DS1_WHO_M_response == 0x3d):
            print("Found LSM9DS1")

    time.sleep(1)


def writeACC(register, value):
    bus.write_byte_data(LSM9DS1_ACC_ADDRESS, register, value)
    return -1


def writeMAG(register, value):
    bus.write_byte_data(LSM9DS1_MAG_ADDRESS, register, value)
```

```
    return -1


def writeGRY(register, value):
    bus.write_byte_data(LSM9DS1_GYR_ADDRESS, register, value)
    return -1


def readACCx():
    acc_l = bus.read_byte_data(LSM9DS1_ACC_ADDRESS, LSM9DS1_OUT_X_L_XL)
    acc_h = bus.read_byte_data(LSM9DS1_ACC_ADDRESS, LSM9DS1_OUT_X_H_XL)
    acc_combined = (acc_l | acc_h << 8)
    return acc_combined if acc_combined < 32768 else acc_combined -
65536


def readACCy():
    acc_l = bus.read_byte_data(LSM9DS1_ACC_ADDRESS, LSM9DS1_OUT_Y_L_XL)
    acc_h = bus.read_byte_data(LSM9DS1_ACC_ADDRESS, LSM9DS1_OUT_Y_H_XL)
    acc_combined = (acc_l | acc_h << 8)
    return acc_combined if acc_combined < 32768 else acc_combined -
65536


def readACCz():
    acc_l = bus.read_byte_data(LSM9DS1_ACC_ADDRESS, LSM9DS1_OUT_Z_L_XL)
    acc_h = bus.read_byte_data(LSM9DS1_ACC_ADDRESS, LSM9DS1_OUT_Z_H_XL)
    acc_combined = (acc_l | acc_h << 8)
    return acc_combined if acc_combined < 32768 else acc_combined -
65536


def readMAGx():
    mag_l = bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_OUT_X_L_M)
    mag_h = bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_OUT_X_H_M)
    mag_combined = (mag_l | mag_h << 8)
    return mag_combined if mag_combined < 32768 else mag_combined -
65536


def readMAGy():
    mag_l = bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_OUT_Y_L_M)
    mag_h = bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_OUT_Y_H_M)
    mag_combined = (mag_l | mag_h << 8)
    return mag_combined if mag_combined < 32768 else mag_combined -
65536


def readMAGz():
    mag_l = bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_OUT_Z_L_M)
    mag_h = bus.read_byte_data(LSM9DS1_MAG_ADDRESS, LSM9DS1_OUT_Z_H_M)
    mag_combined = (mag_l | mag_h << 8)
    return mag_combined if mag_combined < 32768 else mag_combined -
65536
```

```
def readGYRx():
    gyr_l = bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_OUT_X_L_G)
    gyr_h = bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_OUT_X_H_G)
    gyr_combined = (gyr_l | gyr_h << 8)
    return gyr_combined if gyr_combined < 32768 else gyr_combined -
65536


def readGYRy():
    gyr_l = bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_OUT_Y_L_G)
    gyr_h = bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_OUT_Y_H_G)
    gyr_combined = (gyr_l | gyr_h << 8)
    return gyr_combined if gyr_combined < 32768 else gyr_combined -
65536


def readGYRz():
    gyr_l = bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_OUT_Z_L_G)
    gyr_h = bus.read_byte_data(LSM9DS1_GYR_ADDRESS, LSM9DS1_OUT_Z_H_G)
    gyr_combined = (gyr_l | gyr_h << 8)
    return gyr_combined if gyr_combined < 32768 else gyr_combined -
65536


def initIMU():

        # initialise the gyroscope
        writeGRY(LSM9DS1_CTRL_REG4, 0b00111000)  # z, y, x axis enabled
for gyro
        writeGRY(LSM9DS1_CTRL_REG1_G, 0b10111000)  # Gyro ODR = 476Hz,
2000 dps
        writeGRY(LSM9DS1_ORIENT_CFG_G, 0b10111000)  # Swap orientation

        # initialise the accelerometer
        writeACC(LSM9DS1_CTRL_REG5_XL, 0b00111000)  # z, y, x axis
enabled for accelerometer
        writeACC(LSM9DS1_CTRL_REG6_XL, 0b00111000)  # +/- 8g

        # initialise the magnetometer
        writeMAG(LSM9DS1_CTRL_REG1_M, 0b10011100)  # Temp compensation
enabled,Low power mode mode,80Hz ODR
        writeMAG(LSM9DS1_CTRL_REG2_M, 0b01000000)  # +/-12gauss
        writeMAG(LSM9DS1_CTRL_REG3_M, 0b00000000)  # continuous update
        writeMAG(LSM9DS1_CTRL_REG4_M, 0b00000000)  # lower power mode
for Z axis


def get_IMU_data():
    ACCx = '{:.2f}'.format(readACCx()*0.244/1000)
    ACCy = '{:.2f}'.format(readACCy()*0.244/1000)
    ACCz = '{:.2f}'.format(readACCz()*0.244/1000)
    GRYx = '{:.2f}'.format(readGYRx()*gain)
    GRYy = '{:.2f}'.format(readGYRy()*gain)
    GRYz = '{:.2f}'.format(readGYRz()*gain)
```

```
    MAGx = '{:.2f}'.format(readMAGx())
    MAGy = '{:.2f}'.format(readMAGy())
    MAGz = '{:.2f}'.format(readMAGz())

  return ACCx, ACCy, ACCz, GRYx, GRYy, GRYz, MAGx, MAGy, MAGz
```

E. LSM9DS1.PY [81]

LSM9DS1 contains the device registers for the LSM9DS1 inertial measurement unit.

Source Authors: Mark Williams and Peter Peck
Name of File: mpl3115a2.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 7 Jan 2018

Inputs: None
Outputs: None

```
LSM9DS1_MAG_ADDRESS = 0x1E
LSM9DS1_ACC_ADDRESS = 0x6B
LSM9DS1_GYR_ADDRESS = 0x6B


#////////////////////////////////////////
#// LSM9DS1 Accel/Gyro (XL/G) Registers //
#////////////////////////////////////////
LSM9DS1_ACT_THS             = 0x04
LSM9DS1_ACT_DUR             = 0x05
LSM9DS1_INT_GEN_CFG_XL      = 0x06
LSM9DS1_INT_GEN_THS_X_XL    = 0x07
LSM9DS1_INT_GEN_THS_Y_XL    = 0x08
LSM9DS1_INT_GEN_THS_Z_XL    = 0x09
LSM9DS1_INT_GEN_DUR_XL      = 0x0A
LSM9DS1_REFERENCE_G         = 0x0B
LSM9DS1_INT1_CTRL           = 0x0C
LSM9DS1_INT2_CTRL           = 0x0D
LSM9DS1_WHO_AM_I_XG         = 0x0F
LSM9DS1_CTRL_REG1_G         = 0x10
LSM9DS1_CTRL_REG2_G         = 0x11
LSM9DS1_CTRL_REG3_G         = 0x12
LSM9DS1_ORIENT_CFG_G        = 0x13
LSM9DS1_INT_GEN_SRC_G       = 0x14
LSM9DS1_OUT_TEMP_L          = 0x15
LSM9DS1_OUT_TEMP_H          = 0x16
LSM9DS1_STATUS_REG_0        = 0x17
LSM9DS1_OUT_X_L_G           = 0x18
LSM9DS1_OUT_X_H_G           = 0x19
LSM9DS1_OUT_Y_L_G           = 0x1A
LSM9DS1_OUT_Y_H_G           = 0x1B
LSM9DS1_OUT_Z_L_G           = 0x1C
LSM9DS1_OUT_Z_H_G           = 0x1D
LSM9DS1_CTRL_REG4           = 0x1E
LSM9DS1_CTRL_REG5_XL        = 0x1F
LSM9DS1_CTRL_REG6_XL        = 0x20
LSM9DS1_CTRL_REG7_XL        = 0x21
LSM9DS1_CTRL_REG8           = 0x22
LSM9DS1_CTRL_REG9           = 0x23
LSM9DS1_CTRL_REG10          = 0x24
LSM9DS1_INT_GEN_SRC_XL      = 0x26
```

```
LSM9DS1_STATUS_REG_1        = 0x27
LSM9DS1_OUT_X_L_XL          = 0x28
LSM9DS1_OUT_X_H_XL          = 0x29
LSM9DS1_OUT_Y_L_XL          = 0x2A
LSM9DS1_OUT_Y_H_XL          = 0x2B
LSM9DS1_OUT_Z_L_XL          = 0x2C
LSM9DS1_OUT_Z_H_XL          = 0x2D
LSM9DS1_FIFO_CTRL           = 0x2E
LSM9DS1_FIFO_SRC            = 0x2F
LSM9DS1_INT_GEN_CFG_G       = 0x30
LSM9DS1_INT_GEN_THS_XH_G    = 0x31
LSM9DS1_INT_GEN_THS_XL_G    = 0x32
LSM9DS1_INT_GEN_THS_YH_G    = 0x33
LSM9DS1_INT_GEN_THS_YL_G    = 0x34
LSM9DS1_INT_GEN_THS_ZH_G    = 0x35
LSM9DS1_INT_GEN_THS_ZL_G    = 0x36
LSM9DS1_INT_GEN_DUR_G       = 0x37

#////////////////////////////
#// LSM9DS1 Magneto Registers //
#////////////////////////////
LSM9DS1_OFFSET_X_REG_L_M    = 0x05
LSM9DS1_OFFSET_X_REG_H_M    = 0x06
LSM9DS1_OFFSET_Y_REG_L_M    = 0x07
LSM9DS1_OFFSET_Y_REG_H_M    = 0x08
LSM9DS1_OFFSET_Z_REG_L_M    = 0x09
LSM9DS1_OFFSET_Z_REG_H_M    = 0x0A
LSM9DS1_WHO_AM_I_M          = 0x0F
LSM9DS1_CTRL_REG1_M         = 0x20
LSM9DS1_CTRL_REG2_M         = 0x21
LSM9DS1_CTRL_REG3_M         = 0x22
LSM9DS1_CTRL_REG4_M         = 0x23
LSM9DS1_CTRL_REG5_M         = 0x24
LSM9DS1_STATUS_REG_M        = 0x27
LSM9DS1_OUT_X_L_M           = 0x28
LSM9DS1_OUT_X_H_M           = 0x29
LSM9DS1_OUT_Y_L_M           = 0x2A
LSM9DS1_OUT_Y_H_M           = 0x2B
LSM9DS1_OUT_Z_L_M           = 0x2C
LSM9DS1_OUT_Z_H_M           = 0x2D
LSM9DS1_INT_CFG_M           = 0x30
LSM9DS1_INT_SRC_M           = 0x30
LSM9DS1_INT_THS_L_M         = 0x32
LSM9DS1_INT_THS_H_M         = 0x33

#////////////////////////////
#// LSM9DS1 WHO_AM_I Responses //
#////////////////////////////
LSM9DS1_WHO_AM_I_AG_RSP     = 0x68
LSM9DS1_WHO_AM_I_M_RSP      = 0x3D
```

F. ADXL377.PY

adxl377 is the Python object for the adxl377 "high-g" accelerometer. The sensor is analog, so the script is designed for communication with the analog-to-digital converter (TI ADS7828). The TI ADS7828 communicates with the RPi via i2c. Currently, channels 3-7 are unused on the TI ADS7828. Future iterations of the rocket board could make use of these excess channels.

Source Author: Dillon Pierce
Name of File: adxl377.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

Inputs: BusID, Slave Address
Outputs: Acceleration-Gs (x,y,z)

```
from smbus import SMBus
import struct

# Analog Channel Input
channel_0 = 0b000
channel_1 = 0b100
channel_2 = 0b001
channel_3 = 0b101
channel_4 = 0b010
channel_5 = 0b110
channel_6 = 0b011
channel_7 = 0b111

# Voltage Reference
v_ref = 2.5

# Command Byte Values
# Format: SD C2 C1 C0 PD1 PD0 X X
# C2 C1 C0 = Channel Address
sd = 1       # Single-ended input
pd1 = 1      # Internal reference ON
pd0 = 1      # A/D Converter ON
g_value = 0  # Initial G reading

# Accelerometer calibration values
x_min = 1.4368
y_min = 1.5399
z_min = 1.4178
x_max = 1.4642
y_max = 1.6296
z_max = 1.4746

x_mean = (x_max+x_min)/2
y_mean = (y_max+y_min)/2
z_mean = (z_max+z_min)/2

x_step = (x_max-x_min)/2
y_step = (y_max-y_min)/2
```

```
z_step = (z_max-z_min)/2

general_step = 0.0065           # volts/g per documentation

class ADXL377(object):

    def __init__(self, busID, slaveAddr):
        self.__i2c = SMBus(busID)
        self.__slave = slaveAddr

    Max_AD = 4096.0                                  # 12-bit ADC

    def read_channel(self, channel):
        command = self.format_cmd_byte(channel)
        data = self.__i2c.read_i2c_block_data(self.__slave, command, 2)
        raw_value = struct.unpack('>H', struct.pack('>BB', data[0],
data[1]))[0]
        voltage = (raw_value / self.Max_AD) * v_ref
        if channel == channel_0:
            g_value = (voltage - x_mean)/general_step
        if channel == channel_1:
            g_value = (voltage - y_mean)/general_step
        if channel == channel_2:
            g_value = (voltage - z_mean)/general_step

        return g_value

    def format_cmd_byte(self, channel):
        command = channel << 4 | 0x8C
        return command

    def get_accel_values(self):
        x_axis = self.read_channel(channel_0)
        y_axis = self.read_channel(channel_1)
        z_axis = self.read_channel(channel_2)
        return '{:.6f}'.format(x_axis), '{:.6f}'.format(y_axis),
'{:.6f}'.format(z_axis)
```

G. ACCELEROMETER_CALIBRATION.PY

Accelerometer calibration is the script used to obtain the adxl377 accelerometer calibration values. To use the calibration script, first place the accelerometer on a relatively level surface. After initiating the script, slowly rotate the sensor board about all three axes. Once complete, press Ctrl+C to terminate the script. The final maximum and minimum calibration values for each axis will be displayed on the screen. Place these calibration values into the adxl377.py object under the '#Accelerometer calibration values' heading for more accurate acceleration values.

Source Author: Dillon Pierce
Name of File: Accelerometer_Calibration.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

Inputs: None
Outputs: Accelerometer max and min calibration values (x,y,z)

```
import adxl377
import time
from gpiozero import LED
import sys

acc = adxl377.ADXL377(busID=1, slaveAddr=0x48)
check_led = LED(19)

x_min = 3.0
y_min = 3.0
z_min = 3.0
x_max = 0.0
y_max = 0.0
z_max = 0.0
counter = 0.0


print('Calibration Commencing - Slowly rotate accelerometer about each
axis when green light turns on')
time.sleep(2)

while True:
    try:
        counter += 1
        acc_data = acc.get_accel_values()
        x_value = float(acc_data[0])
        y_value = float(acc_data[1])
        z_value = float(acc_data[2])

        if counter > 100:
            print(acc_data)
            check_led.on()
            if x_value > x_max :
                x_max = x_value
```

```
            if x_value < x_min:
                x_min = x_value
            if y_value > y_max:
                y_max = y_value
            if y_value < y_min:
                y_min = y_value
            if z_value > z_max:
                z_max = z_value
            if z_value < z_min:
                z_min = z_value

        print('X Max: {:.4f}'.format(x_max), 'Y Max:
{:.4f}'.format(y_max), 'Z Max: {:.4f}'.format(z_max))
        print('X Min: {:.4f}'.format(x_min), 'Y Min:
{:.4f}'.format(y_min), 'Z Min: {:.4f}'.format(z_min))

    except KeyboardInterrupt:
        print('Final Calibration values:')
        print('X Max: {:.4f}'.format(x_max), 'Y Max:
{:.4f}'.format(y_max), 'Z Max: {:.4f}'.format(z_max))
        print('X Min: {:.4f}'.format(x_min), 'Y Min:
{:.4f}'.format(y_min), 'Z Min: {:.4f}'.format(z_min))
        sys.exit()
```

H. DATA PROCESSING.PY

Data processing is the script used to process the data file generated by the main flight software. To conduct data processing, simply initiate the script and select the data.txt file for processing in the dialog box that appears.

Source Author: Dillon Pierce
Name of File: Data Processing.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: 23 May 2019

Inputs: Data .txt File
Outputs: Various graphs and key flight performance parameters

```
import matplotlib.pyplot as pyplot
from tkinter import Tk
from tkinter import filedialog

pyplot.rcParams['font.sans-serif'] = "Times New Roman"
pyplot.rcParams['font.family'] = "sans-serif"

Tk().withdraw()  # we don't want a full GUI, so keep the root window
from appearing
filename = filedialog.askopenfilename()  # show an "Open" dialog box
and return the path to the selected file
fp = open(filename, 'r')     # Open file in read mode

t = []
ACCx = []
ACCy = []
ACCz = []
accx = []
accy = []
accz = []
GRYx = []
GRYy = []
GRYz = []
MAGx = []
MAGy = []
MAGz = []
temp = []
pres = []
alt = []

fp.readline()

for line in fp.readlines():
    cols = line.split(',')
    if len(t) == 0:
        start_time = float(cols[1])

    t.append(float(cols[1])-start_time)
    ACCx.append(float(cols[2]))
```

```
        ACCy.append(float(cols[3]))
        ACCz.append(float(cols[4]))
        accx.append(float(cols[5]))
        accy.append(float(cols[6]))
        accz.append(float(cols[7]))
        GRYx.append(float(cols[8]))
        GRYy.append(float(cols[9]))
        GRYz.append(float(cols[10]))
        MAGx.append(float(cols[11]))
        MAGy.append(float(cols[12]))
        MAGz.append(float(cols[13]))
        temp.append(float(cols[14]))
        pres.append(float(cols[15]))
        alt_reading = cols[16]
        alt.append(float(alt_reading[:-2]))

# Find and Display Maximum Altitude

start_alt = alt[0]
max_alt = max(alt)
max_alt_ftAGL = (max_alt-start_alt)*3.28084
print('Maximum Altitude (AGL): %d feet' % max_alt_ftAGL)

# Plot Results

pyplot.subplot(131)
pyplot.title('IMU Acceleration vs. Time', fontsize=15, weight='bold')
pyplot.plot(t, ACCx, t, ACCy, t, ACCz, linewidth=1)
pyplot.xlabel('Time (s)')
pyplot.ylabel('Acceleration (Gs)')
pyplot.grid(b=None, which='major', axis='both')
axis_labels = ['X-Axis', 'Y-Axis', 'Z-Axis']
pyplot.legend(axis_labels)

pyplot.subplot(132)
pyplot.title('ADXL377 Acceleration vs. Time', fontsize=15,
weight='bold')
pyplot.plot(t, accx, t, accy, t, accz, linewidth=1)
pyplot.xlabel('Time (s)')
pyplot.ylabel('Acceleration (Gs)')
pyplot.grid(b=None, which='major', axis='both')
axis_labels = ['X-Axis', 'Y-Axis', 'Z-Axis']
pyplot.legend(axis_labels)

pyplot.subplot(133)
pyplot.title('Rotation vs. Time', fontsize=15, weight='bold')
pyplot.plot(t, GRYx, t, GRYy, t, GRYz, linewidth=1)
pyplot.xlabel('Time (s)')
pyplot.ylabel('Rotation Rate (deg/s)')
pyplot.grid(b=None, which='major', axis='both')
axis_labels = ['X-Axis', 'Y-Axis', 'Z-Axis']
pyplot.legend(axis_labels)

pyplot.figure(2)
pyplot.subplot(121)
```

```
pyplot.title('Altitude vs. Time', fontsize=15, weight='bold')
pyplot.grid(b=None, which='major', axis='both')
pyplot.plot(t, alt)
pyplot.xlabel('Time (s)')
pyplot.ylabel('Altitude (m)')


pyplot.subplot(122)
pyplot.title('Temperature vs. Time', fontsize=15, weight='bold')
pyplot.plot(t, temp)
pyplot.grid(b=None, which='major', axis='both')
pyplot.xlabel('Time (s)')
pyplot.ylabel('Temperature (deg C)')
pyplot.show()
```

I. UBLOX.PY [82]

ublox is the Python object for the NEO-M8N series GPS. The sensor communicates with the RPi via SPI. Various GPS messages can be pulled from the device. Currently, the rocket board pulls data from the NAV_POSLLH, NAV_STATUS, NAV_VELNED, and NAV_PVT messages.

Source Author: Andrew Tridgell
Name of File: ublox.py
File Location: https://github.com/dpierce1274/NPS-RocketBoard.git
Date Last Modified: Oct 2012

Inputs: None
Outputs: Various GPS messages


```
import struct
import time, os
import sys

# specify Python version
if sys.version_info[0] < 3: # we're on python 2.x.x
    PYTHON_VERSION = 2
else:
    PYTHON_VERSION = 3

# protocol constants
PREAMBLE1 = 0xb5
PREAMBLE2 = 0x62

# message classes
CLASS_NAV = 0x01
CLASS_RXM = 0x02
CLASS_INF = 0x04
CLASS_ACK = 0x05
CLASS_CFG = 0x06
CLASS_MON = 0x0A
CLASS_AID = 0x0B
CLASS_TIM = 0x0D
CLASS_ESF = 0x10

# ACK messages
MSG_ACK_NACK = 0x00
MSG_ACK_ACK = 0x01

# NAV messages
MSG_NAV_POSECEF    = 0x1
MSG_NAV_POSLLH     = 0x2
MSG_NAV_STATUS     = 0x3
MSG_NAV_DOP        = 0x4
MSG_NAV_SOL        = 0x6
MSG_NAV_POSUTM     = 0x8
```

```
MSG_NAV_VELNED    = 0x12
MSG_NAV_VELECEF   = 0x11
MSG_NAV_TIMEGPS   = 0x20
MSG_NAV_TIMEUTC   = 0x21
MSG_NAV_CLOCK     = 0x22
MSG_NAV_SVINFO    = 0x30
MSG_NAV_AOPSTATUS = 0x60
MSG_NAV_DGPS      = 0x31
MSG_NAV_DOP       = 0x04
MSG_NAV_EKFSTATUS = 0x40
MSG_NAV_SBAS      = 0x32
MSG_NAV_SOL       = 0x06
MSG_NAV_PVT       = 0x07

# RXM messages
MSG_RXM_RAW    = 0x10
MSG_RXM_SFRB   = 0x11
MSG_RXM_SVSI   = 0x20
MSG_RXM_EPH    = 0x31
MSG_RXM_ALM    = 0x30
MSG_RXM_PMREQ  = 0x41

# AID messages
MSG_AID_ALM    = 0x30
MSG_AID_EPH    = 0x31
MSG_AID_ALPSRV = 0x32
MSG_AID_AOP    = 0x33
MSG_AID_DATA   = 0x10
MSG_AID_ALP    = 0x50
MSG_AID_DATA   = 0x10
MSG_AID_HUI    = 0x02
MSG_AID_INI    = 0x01
MSG_AID_REQ    = 0x00

# CFG messages
MSG_CFG_PRT = 0x00
MSG_CFG_ANT = 0x13
MSG_CFG_DAT = 0x06
MSG_CFG_EKF = 0x12
MSG_CFG_ESFGWT = 0x29
MSG_CFG_CFG = 0x09
MSG_CFG_USB = 0x1b
MSG_CFG_RATE = 0x08
MSG_CFG_SET_RATE = 0x01
MSG_CFG_NAV5 = 0x24
MSG_CFG_FXN = 0x0E
MSG_CFG_INF = 0x02
MSG_CFG_ITFM = 0x39
MSG_CFG_MSG = 0x01
MSG_CFG_NAVX5 = 0x23
MSG_CFG_NMEA = 0x17
MSG_CFG_NVS = 0x22
MSG_CFG_PM2 = 0x3B
MSG_CFG_PM = 0x32
MSG_CFG_RINV = 0x34
```

```
MSG_CFG_RST = 0x04
MSG_CFG_RXM = 0x11
MSG_CFG_SBAS = 0x16
MSG_CFG_TMODE2 = 0x3D
MSG_CFG_TMODE = 0x1D
MSG_CFG_TPS = 0x31
MSG_CFG_TP = 0x07
MSG_CFG_GNSS = 0x3E

# ESF messages
MSG_ESF_MEAS   = 0x02
MSG_ESF_STATUS = 0x10

# INF messages
MSG_INF_DEBUG  = 0x04
MSG_INF_ERROR  = 0x00
MSG_INF_NOTICE = 0x02
MSG_INF_TEST   = 0x03
MSG_INF_WARNING= 0x01

# MON messages
MSG_MON_SCHD  = 0x01
MSG_MON_HW    = 0x09
MSG_MON_HW2   = 0x0B
MSG_MON_IO    = 0x02
MSG_MON_MSGPP = 0x06
MSG_MON_RXBUF = 0x07
MSG_MON_RXR   = 0x21
MSG_MON_TXBUF = 0x08
MSG_MON_VER   = 0x04

# TIM messages
MSG_TIM_TP   = 0x01
MSG_TIM_TM2  = 0x03
MSG_TIM_SVIN = 0x04
MSG_TIM_VRFY = 0x06

# port IDs
PORT_DDC    =0
PORT_SERIAL1=1
PORT_SERIAL2=2
PORT_USB    =3
PORT_SPI    =4

# dynamic models
DYNAMIC_MODEL_PORTABLE   = 0
DYNAMIC_MODEL_STATIONARY = 2
DYNAMIC_MODEL_PEDESTRIAN = 3
DYNAMIC_MODEL_AUTOMOTIVE = 4
DYNAMIC_MODEL_SEA        = 5
DYNAMIC_MODEL_AIRBORNE1G = 6
DYNAMIC_MODEL_AIRBORNE2G = 7
DYNAMIC_MODEL_AIRBORNE4G = 8

#reset items
```

```
RESET_HOT  = 0
RESET_WARM = 1
RESET_COLD = 0xFFFF

RESET_HW            = 0
RESET_SW            = 1
RESET_SW_GPS        = 2
RESET_HW_GRACEFUL   = 4
RESET_GPS_STOP      = 8
RESET_GPS_START     = 9

class UBloxError(Exception):
    '''Ublox error class'''
    def __init__(self, msg):
        Exception.__init__(self, msg)
        self.message = msg

class UBloxAttrDict(dict):
    '''allow dictionary members as attributes'''
    def __init__(self):
        dict.__init__(self)

    def __getattr__(self, name):
        try:
            return self.__getitem__(name)
        except KeyError:
            raise AttributeError(name)

    def __setattr__(self, name, value):
        if self.__dict__.has_key(name):
            # allow set on normal attributes
            dict.__setattr__(self, name, value)
        else:
            self.__setitem__(name, value)

def ArrayParse(field):
    '''parse an array descriptor'''
    arridx = field.find('[')
    if arridx == -1:
        return (field, -1)
    alen = int(field[arridx+1:-1])
    fieldname = field[:arridx]
    return (fieldname, alen)

class UBloxDescriptor:
    '''class used to describe the layout of a UBlox message'''
    def __init__(self, name, msg_format, fields=[], count_field=None,
format2=None, fields2=None):
        self.name = name
        self.msg_format = msg_format
        self.fields = fields
        self.count_field = count_field
        self.format2 = format2
        self.fields2 = fields2
```

```
    def getf(self, fmt, buf, size):
            f = list(struct.unpack(fmt, buf[:size]))
            return f

    def unpack(self, msg):
        '''unpack a UBloxMessage, creating the .fields and ._recs
attributes in msg'''
        msg._fields = {}

        # unpack main message blocks. A comm
        formats = self.msg_format.split(',')
        buf = msg._buf[6:-2]
        count = 0
        msg._recs = []
        fields = self.fields[:]

        for fmt in formats:
            size1 = struct.calcsize(fmt)
            if size1 > len(buf):
                raise UBloxError("%s INVALID_SIZE1=%u" % (self.name,
len(buf)))
            f1 = self.getf(fmt, buf, size1)

            i = 0
            while i < len(f1):
                field = fields.pop(0)
                (fieldname, alen) = ArrayParse(field)
                if alen == -1:
                    msg._fields[fieldname] = f1[i]
                    if self.count_field == fieldname:
                        count = int(f1[i])
                    i += 1
                else:
                    msg._fields[fieldname] = [0]*alen
                    for a in range(alen):
                        msg._fields[fieldname][a] = f1[i]
                        i += 1
            buf = buf[size1:]
            if len(buf) == 0:
                break

        if self.count_field == '_remaining':
            count = len(buf) / struct.calcsize(self.format2)

        if count == 0:
            msg._unpacked = True
            if len(buf) != 0:
                raise UBloxError("EXTRA_BYTES=%u" % len(buf))
            return

        size2 = struct.calcsize(self.format2)
        for c in range(count):
            r = UBloxAttrDict()
            if size2 > len(buf):
                raise UBloxError("INVALID_SIZE=%u, " % len(buf))
```

```
            f2 = self.getf(self.format2, buf, size2)

            for i in range(len(self.fields2)):
                r[self.fields2[i]] = f2[i]
            buf = buf[size2:]
            msg._recs.append(r)
        if len(buf) != 0:
            raise UBloxError("EXTRA_BYTES=%u" % len(buf))
        msg._unpacked = True

    def pack(self, msg, msg_class=None, msg_id=None):
        '''pack a UBloxMessage from the .fields and ._recs attributes
in msg'''
        f1 = []
        if msg_class is None:
            msg_class = msg.msg_class()
        if msg_id is None:
            msg_id = msg.msg_id()
        msg._buf = ''

        fields = self.fields[:]
        for f in fields:
            (fieldname, alen) = ArrayParse(f)
            if not fieldname in msg._fields:
                break
            if alen == -1:
                f1.append(msg._fields[fieldname])
            else:
                for a in range(alen):
                    f1.append(msg._fields[fieldname][a])
        try:
            # try full length message
            fmt = self.msg_format.replace(',', '')
            msg._buf = struct.pack(fmt, *tuple(f1))
        except Exception as e:
            # try without optional part
            fmt = self.msg_format.split(',')[0]
            msg._buf = struct.pack(fmt, *tuple(f1))

        length = len(msg._buf)
        if msg._recs:
            length += len(msg._recs) * struct.calcsize(self.format2)
        header = struct.pack('<BBBBH', PREAMBLE1, PREAMBLE2, msg_class,
msg_id, length)
        msg._buf = header + msg._buf

        for r in msg._recs:
            f2 = []
            for f in self.fields2:
                f2.append(r[f])
            msg._buf += struct.pack(self.format2, *tuple(f2))
        msg._buf += struct.pack('<BB',
*msg.checksum(data=msg._buf[2:]))

    def format(self, msg):
```

```
        '''return a formatted string for a message'''
        if not msg._unpacked:
            self.unpack(msg)
        ret = self.name + ': '
        for f in self.fields:
            (fieldname, alen) = ArrayParse(f)
            if not fieldname in msg._fields:
                continue
            v = msg._fields[fieldname]
            if isinstance(v, list):
                ret += '%s=[' % fieldname
                for a in range(alen):
                    ret += '%s, ' % v[a]
                ret = ret[:-2] + '], '
            elif isinstance(v, str):
                ret += '%s="%s", ' % (f, v.rstrip(' \0'))
            else:
                ret += '%s=%s, ' % (f, v)
        for r in msg._recs:
            ret += '[ '
            for f in self.fields2:
                v = r[f]
                ret += '%s=%s, ' % (f, v)
            ret = ret[:-2] + ' ], '
        return ret[:-2]


# list of supported message types.
msg_types = {
    (CLASS_ACK, MSG_ACK_ACK)    : UBloxDescriptor('ACK_ACK',
                                                  '<BB',
                                                  ['clsID', 'msgID']),
    (CLASS_ACK, MSG_ACK_NACK)   : UBloxDescriptor('ACK_NACK',
                                                  '<BB',
                                                  ['clsID', 'msgID']),
    (CLASS_CFG, MSG_CFG_USB)    : UBloxDescriptor('CFG_USB',
                                                  '<HHHHHH32s32s32s',
                                                  ['vendorID',
'productID', 'reserved1', 'reserved2', 'powerConsumption',
                                                   'flags',
'vendorString', 'productString', 'serialNumber']),
    (CLASS_CFG, MSG_CFG_PRT)    : UBloxDescriptor('CFG_PRT',
                                                  '<BBHIIHHHH',
                                                  ['portID',
'reserved0', 'txReady', 'mode', 'baudRate', 'inProtoMask',
                                                   'outProtoMask',
'reserved4', 'reserved5']),
    (CLASS_CFG, MSG_CFG_CFG)    : UBloxDescriptor('CFG_CFG',
                                                  '<III,B',
                                                  ['clearMask',
'saveMask', 'loadMask', 'deviceMask']),
    (CLASS_CFG, MSG_CFG_RST)    : UBloxDescriptor('CFG_RST',
                                                  '<HBB',
                                                  ['navBbrMask ',
'resetMode', 'reserved1']),
```

```
    (CLASS_CFG, MSG_CFG_SBAS)   : UBloxDescriptor('CFG_SBAS',
                                                  '<BBBBI',
                                                  ['mode', 'usage',
'maxSBAS', 'scanmode2', 'scanmode1']),
    (CLASS_CFG, MSG_CFG_GNSS)   : UBloxDescriptor('CFG_GNSS',
                                                  '<BBBBBBBBI',
                                                  ['msgVer',
'numTrkChHw', 'numTrkChUse', 'numConfigBlocks', 'gnssId',
                                                   'resTrkCh',
'maxTrkCh', 'resetved1', 'flags']),
    (CLASS_CFG, MSG_CFG_RATE)   : UBloxDescriptor('CFG_RATE',
                                                  '<HHH',
                                                  ['measRate',
'navRate', 'timeRef']),
    (CLASS_CFG, MSG_CFG_MSG)    : UBloxDescriptor('CFG_MSG',
                                                  '<BB6B',
                                                  ['msgClass', 'msgId',
'rates[6]']),
    (CLASS_NAV, MSG_NAV_POSLLH) : UBloxDescriptor('NAV_POSLLH',
                                                  '<IiiiiII',
                                                  ['iTOW', 'Longitude',
'Latitude', 'height', 'hMSL', 'hAcc', 'vAcc']),
    (CLASS_NAV, MSG_NAV_PVT) : UBloxDescriptor('NAV_PVT',

'<IHBBBBBBIiBBBBiiiiIIiiiiiIIHBBBBBBihH',
                                                  ['iTOW', 'year',
'month', 'day', 'hour', 'min', 'sec', 'valid',
                                                   'tAcc', 'nano',
'fixType', 'flags', 'flags2', 'numSV', 'lon',
                                                   'lat', 'height',
'hMSL', 'hAcc', 'vAcc', 'velN', 'velE', 'velD',
                                                   'gSpeed', 'headMot',
'sAcc', 'headAcc', 'pDOP', 'reserved1',
                                                   'reserved2',
'reserved3','reserved4','reserved5','reserved6',
                                                   'headVeh', 'magDec',
'magAcc']),
    (CLASS_NAV, MSG_NAV_VELNED) : UBloxDescriptor('NAV_VELNED',
                                                  '<IiiiIIiII',
                                                  ['iTOW', 'velN',
'velE', 'velD', 'speed', 'gSpeed', 'heading',
                                                   'sAcc', 'cAcc']),
    (CLASS_NAV, MSG_NAV_DOP)    : UBloxDescriptor('NAV_DOP',
                                                  '<IHHHHHHH',
                                                  ['iTOW', 'gDOP',
'pDOP', 'tDOP', 'vDOP', 'hDOP', 'nDOP', 'eDOP']),
    (CLASS_NAV, MSG_NAV_STATUS) : UBloxDescriptor('NAV_STATUS',
                                                  '<IBBBBII',
                                                  ['iTOW', 'gpsFix',
'flags', 'fixStat', 'flags2', 'ttff', 'msss']),
    (CLASS_NAV, MSG_NAV_SOL)    : UBloxDescriptor('NAV_SOL',
                                                  '<IihBBiiiIiiiIHBBI',
                                                  ['iTOW', 'fTOW',
'week', 'gpsFix', 'flags', 'ecefX', 'ecefY', 'ecefZ',
```

```
                                                    'pAcc', 'ecefVX',
'ecefVY', 'ecefVZ', 'sAcc', 'pDOP', 'reserved1',
                                                    'numSV',
'reserved2']),
    (CLASS_NAV, MSG_NAV_POSUTM) : UBloxDescriptor('NAV_POSUTM',
                                                  '<Iiiibb',
                                                  ['iTOW', 'East',
'North', 'Alt', 'Zone', 'Hem']),
    (CLASS_NAV, MSG_NAV_SBAS)   : UBloxDescriptor('NAV_SBAS',
                                                  '<IBBbBBBBB',
                                                  ['iTOW', 'geo',
'mode', 'sys', 'service', 'cnt', 'reserved01', 'reserved02',
'reserved03' ],
                                                  'cnt',
                                                  'BBBBBBhHh',
                                                  ['svid', 'flags',
'udre', 'svSys', 'svService', 'reserved1',
                                                   'prc', 'reserved2',
'ic']),
    (CLASS_NAV, MSG_NAV_POSECEF): UBloxDescriptor('NAV_POSECEF',
                                                  '<IiiiI',
                                                  ['iTOW', 'ecefX',
'ecefY', 'ecefZ', 'pAcc']),
    (CLASS_NAV, MSG_NAV_VELECEF): UBloxDescriptor('NAV_VELECEF',
                                                  '<IiiiI',
                                                  ['iTOW', 'ecefVX',
'ecefVY', 'ecefVZ', 'sAcc']),
    (CLASS_NAV, MSG_NAV_TIMEGPS): UBloxDescriptor('NAV_TIMEGPS',
                                                  '<IihbBI',
                                                  ['iTOW', 'fTOW',
'week', 'leapS', 'valid', 'tAcc']),
    (CLASS_NAV, MSG_NAV_TIMEUTC): UBloxDescriptor('NAV_TIMEUTC',
                                                  '<IIiHBBBBBB',
                                                  ['iTOW', 'tAcc',
'nano', 'year', 'month', 'day', 'hour', 'min', 'sec', 'valid']),
    (CLASS_NAV, MSG_NAV_CLOCK)  : UBloxDescriptor('NAV_CLOCK',
                                                  '<IiiII',
                                                  ['iTOW', 'clkB',
'clkD', 'tAcc', 'fAcc']),
    (CLASS_NAV, MSG_NAV_DGPS)   : UBloxDescriptor('NAV_DGPS',
                                                  '<IihhBBH',
                                                  ['iTOW', 'age',
'baseId', 'baseHealth', 'numCh', 'status', 'reserved1'],
                                                  'numCh',
                                                  '<BBHff',
                                                  ['svid', 'flags',
'ageC', 'prc', 'prrc']),
    (CLASS_NAV, MSG_NAV_SVINFO) : UBloxDescriptor('NAV_SVINFO',
                                                  '<IBBH',
                                                  ['iTOW', 'numCh',
'globalFlags', 'reserved2'],
                                                  'numCh',
                                                  '<BBBBBbhi',
                                                  ['chn', 'svid',
'flags', 'quality', 'cno', 'elev', 'azim', 'prRes']),
```

```
    (CLASS_RXM, MSG_RXM_SVSI)   : UBloxDescriptor('RXM_SVSI',
                                                  '<IhBB',
                                                  ['iTOW', 'week',
'numVis', 'numSV'],
                                                  'numSV',
                                                  '<BBhbB',
                                                  ['svid', 'svFlag',
'azim', 'elev', 'age']),
    (CLASS_RXM, MSG_RXM_EPH)    : UBloxDescriptor('RXM_EPH',
                                                  '<II , 8I 8I 8I',
                                                  ['svid', 'how',
                                                   'sf1d[8]',
'sf2d[8]', 'sf3d[8]']),
    (CLASS_AID, MSG_AID_EPH)    : UBloxDescriptor('AID_EPH',
                                                  '<II , 8I 8I 8I',
                                                  ['svid', 'how',
                                                   'sf1d[8]',
'sf2d[8]', 'sf3d[8]']),
    (CLASS_AID, MSG_AID_AOP)    : UBloxDescriptor('AID_AOP',
                                                  '<B47B , 48B 48B
48B',
                                                  ['svid', 'data[47]',
'optional0[48]', 'optional1[48]', 'optional1[48]']),
    (CLASS_RXM, MSG_RXM_RAW)   : UBloxDescriptor('RXM_RAW',
                                                  '<ihBB',
                                                  ['iTOW', 'week',
'numSV', 'reserved1'],
                                                  'numSV',
                                                  '<ddfBbbB',
                                                  ['cpMes', 'prMes',
'doMes', 'sv', 'mesQI', 'cno', 'lli']),
    (CLASS_RXM, MSG_RXM_SFRB)  : UBloxDescriptor('RXM_SFRB',
                                                  '<BB10I',
                                                  ['chn', 'svid',
'dwrd[10]']),
    (CLASS_AID, MSG_AID_ALM)   : UBloxDescriptor('AID_ALM',
                                                  '<II',
                                                 '_remaining',
                                                 'I',
                                                 ['dwrd']),
    (CLASS_RXM, MSG_RXM_ALM)   : UBloxDescriptor('RXM_ALM',
                                                  '<II , 8I',
                                                  ['svid', 'week',
'dwrd[8]']),
    (CLASS_CFG, MSG_CFG_NAV5)   : UBloxDescriptor('CFG_NAV5',
                                                  '<HBBiIbBHHHHBBIII',
                                                  ['mask', 'dynModel',
'fixMode', 'fixedAlt', 'fixedAltVar', 'minElev',
                                                   'drLimit', 'pDop',
'tDop', 'pAcc', 'tAcc', 'staticHoldThresh',
                                                   'dgpsTimeOut',
'reserved2', 'reserved3', 'reserved4']),
    (CLASS_CFG, MSG_CFG_NAVX5)   : UBloxDescriptor('CFG_NAVX5',

'<HHIBBBBBBBBBBHIBBBBBBBHII',
```

```
                                                ['version', 'mask1',
'reserved0', 'reserved1', 'reserved2',
                                                 'minSVs', 'maxSVs',
'minCNO', 'reserved5', 'iniFix3D',
                                                 'reserved6',
'reserved7', 'reserved8', 'wknRollover',
                                                 'reserved9',
'reserved10', 'reserved11',
                                                 'usePPP', 'useAOP',
'reserved12', 'reserved13',
                                                 'aopOrbMaxErr',
'reserved3', 'reserved4']),
    (CLASS_MON, MSG_MON_HW)     : UBloxDescriptor('MON_HW',
'<IIIIHHBBBBIB25BHIII',
                                                ['pinSel', 'pinBank',
'pinDir', 'pinVal', 'noisePerMS', 'agcCnt', 'aStatus',
                                                 'aPower', 'flags',
'reserved1', 'usedMask',
                                                 'VP[25]',
                                                 'jamInd',
'reserved3', 'pinInq',
                                                 'pullH', 'pullL']),
    (CLASS_MON, MSG_MON_HW2)    : UBloxDescriptor('MON_HW2',
                                                '<bBbBB3BI8BI4B',
                                                ['ofsI', 'magI',
'ofsQ', 'magQ', 'cfgSource', 'reserved1[3]',
                                                 'lowLevCfg',
'reserved2[8]', 'postStatus', 'reserved3[4]']),
    (CLASS_MON, MSG_MON_SCHD)   : UBloxDescriptor('MON_SCHD',
                                                '<IIIIHHHBB',
                                                ['tskRun', 'tskSchd',
'tskOvrr', 'tskReg', 'stack',
                                                 'stackSize',
'CPUIdle', 'flySly', 'ptlSly']),
    (CLASS_MON, MSG_MON_VER)    : UBloxDescriptor('MON_VER',
                                                '<30s10s,30s',
                                                ['swVersion',
'hwVersion', 'romVersion'],
                                                '_remaining',
                                                '30s',
                                                ['extension']),
    (CLASS_TIM, MSG_TIM_TP)     : UBloxDescriptor('TIM_TP',
                                                '<IIiHBB',
                                                ['towMS', 'towSubMS',
'qErr', 'week', 'flags', 'reserved1']),
    (CLASS_TIM, MSG_TIM_TM2)    : UBloxDescriptor('TIM_TM2',
                                                '<BBHHHIIIII',
                                                ['ch', 'flags',
'count', 'wnR', 'wnF', 'towMsR', 'towSubMsR',
                                                 'towMsF',
'towSubMsF', 'accEst']),
    (CLASS_TIM, MSG_TIM_SVIN)   : UBloxDescriptor('TIM_SVIN',
                                                '<IiiiIIBBH',
```

```
                                                    ['dur', 'meanX',
'meanY', 'meanZ', 'meanV',
                                                     'obs', 'valid',
'active', 'reserved1']),
    (CLASS_INF, MSG_INF_ERROR)  : UBloxDescriptor('INF_ERR', '<18s',
['str']),
    (CLASS_INF, MSG_INF_DEBUG)  : UBloxDescriptor('INF_DEBUG', '<18s',
['str'])
}


class UBloxMessage:
    '''UBlox message class - holds a UBX binary message'''
    def __init__(self):
        self._buf = b""
        self._fields = {}
        self._recs = []
        self._unpacked = False
        self.debug_level = 0

    def __str__(self):
        '''format a message as a string'''
        if not self.valid():
            return 'UBloxMessage(INVALID)'
        type = self.msg_type()
        if type in msg_types:
            return msg_types[type].format(self)
        return 'UBloxMessage(UNKNOWN %s, %u)' % (str(type),
self.msg_length())

    def __getattr__(self, name):
        '''allow access to message fields'''
        try:
            return self._fields[name]
        except KeyError:
            if name == 'recs':
                return self._recs
            raise AttributeError(name)

    def __setattr__(self, name, value):
        '''allow access to message fields'''
        if name.startswith('_'):
            self.__dict__[name] = value
        else:
            self._fields[name] = value

    def have_field(self, name):
        '''return True if a message contains the given field'''
        return name in self._fields

    def debug(self, level, msg):
        '''write a debug message'''
        if self.debug_level >= level:
            print(msg)
```

```
    def unpack(self):
        '''unpack a message'''
        if not self.valid():
            raise UBloxError('INVALID MESSAGE')
        type = self.msg_type()
        if not type in msg_types:
            raise UBloxError('Unknown message %s length=%u' %
(str(type), len(self._buf)))
        msg_types[type].unpack(self)

    def pack(self):
        '''pack a message'''
        if not self.valid():
            raise UbloxError('INVALID MESSAGE')
        type = self.msg_type()
        if not type in msg_types:
            raise UBloxError('Unknown message %s' % str(type))
        msg_types[type].pack(self)

    def name(self):
        '''return the short string name for a message'''
        if not self.valid():
            raise UbloxError('INVALID MESSAGE')
        type = self.msg_type()
        if not type in msg_types:
            raise UBloxError('Unknown message %s length=%u' %
(str(type), len(self._buf)))
        return msg_types[type].name

    if PYTHON_VERSION == 2:
        def msg_class(self):
            '''return the message class'''
            return ord(self._buf[2])

        def msg_id(self):
            '''return the message id within the class'''
            return ord(self._buf[3])
    else:
        def msg_class(self):
            '''return the message class'''
            return (self._buf[2])

        def msg_id(self):
            '''return the message id within the class'''
            return (self._buf[3])

    def msg_type(self):
        '''return the message type tuple (class, id)'''
        return (self.msg_class(), self.msg_id())

    def msg_length(self):
        '''return the payload length'''
        (payload_length,) = struct.unpack('<H', self._buf[4:6])
        return payload_length
```

```
    def valid_so_far(self):
        '''check if the message is valid so far'''
        if PYTHON_VERSION == 2:
            if len(self._buf) > 0 and ord(self._buf[0]) != PREAMBLE1:
                return False
            if len(self._buf) > 1 and ord(self._buf[1]) != PREAMBLE2:
                self.debug(1, "bad pre2")
                return False
        else:
            if len(self._buf) > 0 and (self._buf[0]) != PREAMBLE1:
                return False
            if len(self._buf) > 1 and (self._buf[1]) != PREAMBLE2:
                self.debug(1, "bad pre2")
                return False

        if self.needed_bytes() == 0 and not self.valid():
            if len(self._buf) > 8:
                self.debug(1, "bad checksum len=%u needed=%u" %
(len(self._buf), self.needed_bytes()))
            else:
                self.debug(1, "bad len len=%u needed=%u" %
(len(self._buf), self.needed_bytes()))
            return False
        return True

    def add(self, bytes):
        '''add some bytes to a message'''

        self._buf += bytes
        while not self.valid_so_far() and len(self._buf) > 0:
            '''handle corrupted streams'''
            self._buf = self._buf[1:]
        if self.needed_bytes() < 0:
            self._buf = b""

    def checksum(self, data=None):
        '''return a checksum tuple for a message'''
        if data is None:
            data = self._buf[2:-2]
        cs = 0
        ck_a = 0
        ck_b = 0
        for i in data:
            if type(i) is str:

                ck_a = (ck_a + ord(i)) & 0xFF
            else:
                ck_a = (ck_a + i) & 0xFF
            ck_b = (ck_b + ck_a) & 0xFF
        return (ck_a, ck_b)

    def valid_checksum(self):
        '''check if the checksum is OK'''
        (ck_a, ck_b) = self.checksum()
        d = self._buf[2:-2]
```

```
        (ck_a2, ck_b2) = struct.unpack('<BB', self._buf[-2:])
        return ck_a == ck_a2 and ck_b == ck_b2

    def needed_bytes(self):
        '''return number of bytes still needed'''
        if len(self._buf) < 6:
            return 8 - len(self._buf)
        return self.msg_length() + 8 - len(self._buf)

    def valid(self):
        '''check if a message is valid'''
        return len(self._buf) >= 8 and self.needed_bytes() == 0 and
self.valid_checksum()


class UBlox:
    '''main UBlox control class.

    port can be a file (for reading only) or a serial device
    '''
    def __init__(self, port, baudrate=115200, timeout=0):

        self.serial_device = port
        self.baudrate = baudrate
        self.use_sendrecv = False
        self.read_only = False
        self.use_xfer = False
        self.debug_level = 0

        if self.serial_device.startswith("tcp:"):
            import socket
            a = self.serial_device.split(':')
            destination_addr = (a[1], int(a[2]))
            self.dev = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.dev.connect(destination_addr)
            self.dev.setblocking(1)
            self.dev.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)
            self.use_sendrecv = True
        elif os.path.isfile(self.serial_device):
            self.read_only = True
            self.dev = open(self.serial_device, mode='rb')
        if self.serial_device.startswith("spi:"):
            import spidev
            bus, cs = map(int,
self.serial_device.split(':')[1].split('.'))
            #print(bus, cs)
            self.use_xfer = True
            self.dev = spidev.SpiDev()
            self.dev.open(bus, cs)
            #We reuse baudrate parameter but it's difficult to get
default paramaters right. So it's better to specify them explicitly
            self.dev.max_speed_hz = baudrate
        else:
            import serial
```

```
            self.dev = serial.Serial(self.serial_device,
baudrate=self.baudrate,
                                     dsrdtr=False, rtscts=False,
xonxoff=False, timeout=timeout)
        self.logfile = None
        self.log = None
        self.preferred_dynamic_model = None
        self.preferred_usePPP = None
        self.preferred_dgps_timeout = None

    def close(self):
        '''close the device'''
        self.dev.close()
        self.dev = None

    def set_debug(self, debug_level):
        '''set debug level'''
        self.debug_level = debug_level

    def debug(self, level, msg):
        '''write a debug message'''
        if self.debug_level >= level:
            print(msg)

    def set_logfile(self, logfile, append=False):
        '''setup logging to a file'''
        if self.log is not None:
            self.log.close()
            self.log = None
        self.logfile = logfile
        if self.logfile is not None:
            if append:
                mode = 'ab'
            else:
                mode = 'wb'
            self.log = open(self.logfile, mode=mode)

    def set_preferred_dynamic_model(self, model):
        '''set the preferred dynamic model for receiver'''
        self.preferred_dynamic_model = model
        if model is not None:
            self.configure_poll(CLASS_CFG, MSG_CFG_NAV5)

    def set_preferred_dgps_timeout(self, timeout):
        '''set the preferred DGPS timeout for receiver'''
        self.preferred_dgps_timeout = timeout
        if timeout is not None:
            self.configure_poll(CLASS_CFG, MSG_CFG_NAV5)

    def set_preferred_usePPP(self, usePPP):
        '''set the preferred usePPP setting for the receiver'''
        if usePPP is None:
            self.preferred_usePPP = None
            return
        self.preferred_usePPP = int(usePPP)
```

```
        self.configure_poll(CLASS_CFG, MSG_CFG_NAVX5)

    def nmea_checksum(self, msg):
        d = msg[1:]
        cs = 0
        for i in d:
            cs ^= ord(i)
        return cs

    def write(self, buf):
        '''write some bytes'''
        if not self.read_only:
            if self.use_sendrecv:
                return self.dev.send(buf)
            elif self.use_xfer:
                spiBuf = [] # form buf
                for b in buf:
                    if type(b) is str:

                        spiBuf.append(ord(b))
                    else:
                        spiBuf.append(b)
                return self.dev.xfer2(spiBuf)
            return self.dev.write(buf)

    def read(self, n):
        '''read some bytes'''

        if self.use_sendrecv:
            import socket
            try:
                buf = self.dev.recv(n)
                return buf
            except socket.error as e:
                return b''
        if self.use_xfer:
            buf = self.dev.readbytes(n)
            return buf

        buf = self.dev.read(n)
        return buf

    def send_nmea(self, msg):
        if not self.read_only:
            s = msg + "*%02X" % self.nmea_checksum(msg)

            if PYTHON_VERSION == 2:
                b = bytearray()
                b.extend(s)
            else:
                b = bytearray()
                b.extend(map(ord, s))
            self.write(b)

    def set_binary(self):
```

```
        '''put a UBlox into binary mode using a NMEA string'''
        if not self.read_only:
            print("try set binary at %u" % self.baudrate)
            self.send_nmea("$PUBX,41,0,0007,0001,%u,0" % self.baudrate)
            self.send_nmea("$PUBX,41,1,0007,0001,%u,0" % self.baudrate)
            self.send_nmea("$PUBX,41,2,0007,0001,%u,0" % self.baudrate)
            self.send_nmea("$PUBX,41,3,0007,0001,%u,0" % self.baudrate)
            self.send_nmea("$PUBX,41,4,0007,0001,%u,0" % self.baudrate)
            self.send_nmea("$PUBX,41,5,0007,0001,%u,0" % self.baudrate)

    def seek_percent(self, pct):
        '''seek to the given percentage of a file'''
        self.dev.seek(0, 2)
        filesize = self.dev.tell()
        self.dev.seek(pct*0.01*filesize)

    def special_handling(self, msg):
        '''handle automatic configuration changes'''
        if msg.name() == 'CFG_NAV5':
            msg.unpack()
            sendit = False
            pollit = False
            if self.preferred_dynamic_model is not None and
msg.dynModel != self.preferred_dynamic_model:
                msg.dynModel = self.preferred_dynamic_model
                sendit = True
                pollit = True
            if self.preferred_dgps_timeout is not None and
msg.dgpsTimeOut != self.preferred_dgps_timeout:
                msg.dgpsTimeOut = self.preferred_dgps_timeout
                self.debug(2, "Setting dgpsTimeOut=%u" %
msg.dgpsTimeOut)
                sendit = True
                # we don't re-poll for this one, as some receivers
refuse to set it
            if sendit:
                msg.pack()
                self.send(msg)
                if pollit:
                    self.configure_poll(CLASS_CFG, MSG_CFG_NAV5)
        if msg.name() == 'CFG_NAVX5' and self.preferred_usePPP is not
None:
            msg.unpack()
            if msg.usePPP != self.preferred_usePPP:
                msg.usePPP = self.preferred_usePPP
                msg.mask = 1<<13
                msg.pack()
                self.send(msg)
                self.configure_poll(CLASS_CFG, MSG_CFG_NAVX5)

    def receive_message_nonblocking(self, seconds=5):
        '''nonblocking receive of one ublox message'''
        with Timeout(seconds=seconds):
            return self.receive_message()
```

```
    def receive_message(self, ignore_eof=False):
        '''blocking receive of one ublox message'''
        msg = UBloxMessage()
        while True:
            n = msg.needed_bytes()
            b = self.read(n)
            if not b:
                if ignore_eof:
                    time.sleep(0.01)
                    continue
                return None
            if self.use_xfer:
                if PYTHON_VERSION == 3:
                    bb = bytearray()
                    for c in b:
                        bb.append(c)
                    b = bb
                else:
                    b = "".join([chr(c) for c in b]) # here str
            msg.add(b)
            if self.log is not None:
                self.log.write(b)
                self.log.flush()
            if msg.valid():
                self.special_handling(msg)
                return msg

    def receive_message_noerror(self, ignore_eof=False):
        '''blocking receive of one ublox message, ignoring errors'''
        try:
            return self.receive_message(ignore_eof=ignore_eof)
        except UBloxError as e:
            print(e)
            return None
        except OSError as e:
            # Occasionally we get hit with 'resource temporarily
unavailable'
            # messages here on the serial device, catch them too.
            print(e)
            return None

    def send(self, msg):
        '''send a preformatted ublox message'''
        if not msg.valid():
            self.debug(1, "invalid send")
            return
        if not self.read_only:
            self.write(msg._buf)

    def send_message(self, msg_class, msg_id, payload):
        '''send a ublox message with class, id and payload'''
        msg = UBloxMessage()
        msg._buf = struct.pack('<BBBBH', 0xb5, 0x62, msg_class, msg_id,
len(payload))
```

```
        msg._buf += payload

        (ck_a, ck_b) = msg.checksum(msg._buf[2:])
        msg._buf += struct.pack('<BB', ck_a, ck_b)
        self.send(msg)

    def configure_solution_rate(self, rate_ms=200, nav_rate=1,
timeref=0):
        '''configure the solution rate in milliseconds'''
        payload = struct.pack('<HHH', rate_ms, nav_rate, timeref)
        self.send_message(CLASS_CFG, MSG_CFG_RATE, payload)

    def configure_message_rate(self, msg_class, msg_id, rate):
        '''configure the message rate for a given message'''
        payload = struct.pack('<BBB', msg_class, msg_id, rate)
        self.send_message(CLASS_CFG, MSG_CFG_SET_RATE, payload)

    def configure_port(self, port=1, inMask=3, outMask=3, mode=2240,
baudrate=None):
        '''configure a IO port'''
        if baudrate is None:
            baudrate = self.baudrate
        payload = struct.pack('<BBHIIHHHH', port, 0xff, 0, mode,
baudrate, inMask, outMask, 0xFFFF, 0xFFFF)
        self.send_message(CLASS_CFG, MSG_CFG_PRT, payload)

    def configure_loadsave(self, clearMask=0, saveMask=0, loadMask=0,
deviceMask=0):
        '''configure configuration load/save'''
        payload = struct.pack('<IIIB', clearMask, saveMask, loadMask,
deviceMask)
        self.send_message(CLASS_CFG, MSG_CFG_CFG, payload)

    def configure_poll(self, msg_class, msg_id, payload=b''):
        '''poll a configuration message'''
        self.send_message(msg_class, msg_id, payload)

    def configure_poll_port(self, portID=None):
        '''poll a port configuration'''
        if portID is None:
            self.configure_poll(CLASS_CFG, MSG_CFG_PRT)
        else:
            self.configure_poll(CLASS_CFG, MSG_CFG_PRT,
struct.pack('<B', portID))

    def configure_min_max_sats(self, min_sats=4, max_sats=32):
        '''Set the minimum/maximum number of satellites for a solution
in the NAVX5 message'''
        payload = struct.pack('<HHIBBBBBBBBBBHIBBBBBBHII', 0, 4, 0, 0,
0, min_sats, max_sats, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0)
        self.send_message(CLASS_CFG, MSG_CFG_NAVX5, payload)

    def module_reset(self, set, mode):
        ''' Reset the module for hot/warm/cold start'''
```

```
        payload = struct.pack('<HBB', set, mode, 0)
        self.send_message(CLASS_CFG, MSG_CFG_RST, payload)

class TimeoutError(Exception):
    pass

import signal
class Timeout:
    def __init__(self, seconds=1, msg='Timeout'):
        self.seconds = seconds
        self.msg = msg
    def handle_timeout(self, signum, frame):
        raise TimeoutError(self.msg)
    def __enter__(self):
        signal.signal(signal.SIGALRM, self.handle_timeout)
        signal.alarm(self.seconds)
    def __exit__(self, type, value, traceback):
        signal.alarm(0)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. PRE-LAUNCH CHECK AND PACKING LISTS

Author: Dillon Pierce
Date: 23 May 2019

Revision Author:
Revision Number:
Revision Date:

A. CHECK LIST

1. L-72 Hours: Make Preparations

Task	POC	Initial When Complete, Date/Time
Conduct Flight Readiness Review		
Distribute Recall Roster		
Conduct HAB HUB analysis		
Locate and test power inverter in vehicle		
Review launch day procedures		
Replace SPOT Gen3 Batteries		
Charge BigRedBee Batteries		
Charge TeleMega Batteries		
Charge GoPros (3 x session, 3 x various)		
Call John Newman to coordinate propellant delivery and facility occupation		

2. L-48 Hours: Pack

Task	POC	Initial When Complete, Date/Time
Pack everything on packing list		
Review launch day procedures		
Conduct HAB HUB analysis		
Create group text		

3. L-24 Hours: Travel

Team Member	Method	4WD	Location

4. L-24 Hours: Launch Setup

Task	POC	Initial When Complete, Date/Time
Call John Newman (661-301-3834) to coordinate propellant delivery and facility occupation		
Setup Launch Rails		
Review launch day procedures and assigned roles		
Conduct HAB HUB analysis to determine launch time		
Setup ground station table		
Ensure SPOT Gen3 trackers in chase vehicles		
Setup launch equipment		
Construct Rocket Motor		
Verify Flight Computer Configurations ☐ Verify primary TeleMega (S/N:) o Drogue Deploy: Apogee o Apogee Lockout: 10 sec o Main Deploy: 450m o Tx Freq.: 434.550 MHz o Callsign: K6NPS o Telemetry Rate: 38400 baud ☐ Verify backup TeleMega (S/N:) o Drogue Deploy: Apogee + 3 sec o Apogee Lockout: 10 sec o Main Deploy: 400m o Tx Freq.: Not Enabled ☐ Verify TeleDongle (S/N:) o Freq: 434.55 MHz o Telemetry Rate: 38400 baud ☐ Verify BigRedBee o Freq: 433.92 MHz o Tx Interval: 5 sec o Callsign: K6NPS -5 o Verify packet Tx and Rx		
Build Deployment Charges ☐ Check resistance of each canister ☐ Make main black powder charges - label		

o ___g – primary o ___g – backup □ Construct Drogue ejection systems o Completely remove the plastic protective sheath from over the initiator head o Cut the plastic protective sheath to about 3/8th inch and re-install on the initiator o Use the cotton tipped applicator with silicon lube to wipe a residue of lube to the inside cavity of the charge cup o Install and pull initiator to within an inch or so of the charge cup o Mix a small amount of 5 min epoxy. Dab a small amount of this epoxy completely around the bottom of the initiator protective sheath o Assemble and let cure for at least 10 minutes o Measure out .2cc of black powder with measuring scoop o Pour powder into the prepared charge cup o Apply a Pyro charge cover disk or use a piece of 3M blue masking tape over the charge cup to seal in the propellant o Carefully place the Charge Cup into the Pyro Housing o Carefully place the spring onto the Puncture Piston o Insert the Puncture Piston into the Pyro Housing o Attach Pyro Housing to the Mounting Cap		

o Repeat process for backup system o Screw 45g CO_2 cartridge into primary ejection system o Screw 75g CO_2 cartridge into backup ejection system [66]		
☐ Assemble the rocket drogue parachute o Place Nomex parachute protector on shock chord o Place SPOT3 tracker on shock chord o Connect chute to shock chord with quick link o Fold the parachute o Place parachute in parachute protector and wrap o Connect shock chord to motor ring o Ensure motor ring is tight o Slide drogue parachute into main body tube o Leave electronics bay quick link out for later connection		
☐ Assemble the electronics bay sled o Place LiPo batteries in mounts o Mount backup and primary flight computers o Ensure switches are open o Connect primary flight computer to switch o Connect backup flight computer to switch o Connect batteries o Close switches o Check functionality of both flight computers o Open switches		

5. Launch Day

REPORT COMPLETED TASKS TO LIST MANAGER:_____

Time	Task	Initial when complete w/ time
L-120	Insert motor into rocket □ Liberally grease casing □ Ensure screwed in tight	
L-110	Assemble electronics bay □ Ensure switches are open □ Connect main parachute BP charges to terminals □ Thread CO_2 leads through bulkhead □ Slide electronics sled onto threaded rods □ Ensure appropriate height of sled □ Connect BP charges to appropriate flight computer □ Connect CO_2 charges to appropriate flight computer □ Ensure continuity of charges □ Place GoPro in mount – Tape for tight fit □ Turn on GoPro – LEAVE ON and START RECORDING ASAP! □ Place forward and rear bulkhead on coupler □ Ensure wingnuts and hex nuts are tight	
L-95	Connect electronics bay to drogue parachute □ Slide SPOT3 onto shock cord □ Turn on SPOT3 □ Liberally grease lower portion of electronics bay □ Slide e-bay into main airframe □ Place 4 x 4-40 shear pins in electronics bay connection	
L-90	Assemble the rocket main parachute □ Place Nomex parachute protector on shock chord □ Connect chute to shock chord with quick link □ Fold the parachute □ Place parachute in parachute protector and wrap □ Connect shock cord to upper airframe	

	□ Slide main parachute into upper airframe tube □ Connect shock cord to electronics bay □ Liberally grease upper portion of electronics bay □ Slide upper airframe onto electronics bay □ Secure upper airframe to electronics bay with 4 x 4-40 retention screws	
L-90	Get final HAB HUB prediction. Confirm Launch time with RSO	
L-75	All Team meeting	
L-75	Turn on Go-Pro for launch time-lapse	
L-60	Set up tables, unload equipment	
L-45	Prep Chase Vehicles with chow and water	
L-30	Set up ground stations x2 □ Connect Chase laptops to power □ Connect MHX radios to Laptops □ Setup MHX car antenna □ Setup AltOS ground station	
L-30	Prepare payload for encapsulation	
L-30	Prepare Nose cone for launch □ Connect ¼" threaded rod to end of nose cone □ Place retention nut at appropriate height □ Connect BigRedBee to battery □ Connect RPi to battery □ Slide sled and rear bulk plate into nose cone □ Secure assembly with eyebolt □ Ensure functionality of all sensors □ Connect nose cone parachute assembly to eyebolt □ Turn on SPOTTrace □ Slide nose cone onto upper airframe □ Secure nose cone with 4 x 6-32 screws	
L-15	Power up Ground Stations □ Turn on laptop □ Run COSMOS "HAB" icon □ Open applicable windows o Command and Telemetry Server o Command Sender o Telemetry Viewer □ Run AltOS	
L-10	Encapsulate payload at the hut □ Ensure rocket CG location: 4" **in front** of CP	

L-5	Prepare for launch ☐ Walk to launch pad ☐ Load Rocket ☐ Slide ignitor all the way into rocket motor ☐ Tape ignitor lead to edge of nozzle – DO NOT PUT ON CAP ☐ Go vertical with rocket ☐ Verify nose cone sensors are still operational ☐ Start Go-Pro recordings o 1 x Go-Pro in rocket – ensure recording o 2 x Go-Pro Launch pad ☐ Turn on primary flight computer o Verify voltage:_____ o Verify flight mode:_____ o Verify continuity:_____ ☐ Turn on backup flight computer o Verify voltage:_____ o Verify flight mode:_____ o Verify continuity:_____ ☐ Discharge alligator clips from launch pad ☐ Verify no charge on alligator clips with meter o Voltage:_____ ☐ Connect alligator clips to ignitor ☐ Verify continuity of ignitor ☐ Excute Main.py on Rocketboard o Background the script ☐ Walk to launch positions	
L-1	Go/No go for launch ☐ Verify telemetry from Rocketboard ☐ Verify telemetry from TeleMega ☐ Verify telemetry from BigRedBee ☐ Place key in control box ☐ Turn control box on	

	□ Confirm NPS rocket Go for launch □ Confirm payload Go for launch □ Visually confirm airspace and surrounding area clear □ Conduct 10 second countdown	
L	Launch □ Time: _____ □ Lat/Long: _____	
L+1	Track rocket via BigRedBee APRS transmission and COSMOS □ Location:	
L+2	Track rocket with TeleMega via ground station □ Location:	
L+15	Breakdown Chase 1 Ground Station for mobile □ Disconnect Chase 1 Laptop and MHX □ Connect MHX to car antenna □ Connect laptop and MHX to inverter □ Power on MHX	
TBD	NPS Rocket Recovery, Team 1 Depart □ Take Shovel □ Take Hand-held GPS □ 1 x IC-T22A □ Water □ Chow □ Time: _____ □ Estimated Destination Lat/Lon: - _____	
TBD	NPS Rocket Recovery, Team 1, Finds Rocket Body	

	☐ Time: _____ ☐ From a distance, verify all ejection charges have ignited ☐ Disarm primary and backup flight computers	
TBD	NPS Rocket Recovery, Team 1, Begins return to FAR ☐ Time: _____	

B. PACKING LIST

1. Nose Cone

- ☐ Nose cone
- ☐ ¼” x 24” threaded aluminum rod
- ☐ ¼” washer
- ☐ Nose cone sled
  - BigRedBee Tracker
    - BigRedBee LiPo Battery
    - 3 x 4/40 mounting screws
  - Li-Ion Battery Pack
  - Raspberry Pi 3 A+
    - SD Card
  - RocketBoard
    - 4 x standoffs
    - 4 x mounting screws
- ☐ ¼” steel eyebolt
- ☐ 2 x quick links
- ☐ SPOTTrace
  - Retention Bag

2. Upper Airframe

- ☐ 4 x 4-40 Nylon shear pins
- ☐ 4 x 6-32 shortened aluminum retention screws
- ☐ Drogue Parachute (24” Fruity Chute)
  - Parachute protector
- ☐ 2 x quick links
- ☐ 30’ Nylon Shock Chord
  - 2 x quick links
  - Shock Chord protector
- ☐ 14” fiberglass section

3. Electronics Bay

- ☐ Forward closure
- ☐ 2 x wing nuts
- ☐ Fiberglass Coupler
- ☐ Electronics Bay Sled
- ☐ 2 x FingerTech switches
  - 4 x 2-56 mounting screws
- ☐ 2 x TeleMega Flight Computers
  - 8 x 4-40 mounting screws
  - 2 x 900mAh LiPo batteries

- Rear Closure
  - 2 x washers
  - 2 x bolts
- 3 x zip ties for mounting
- 2 x RAPTOR Ejection Systems
  - 45g CO_2 Cartridge
  - 75g CO_2 Cartridge
  - Pyro charge cover disks
  - Pyro housing
  - Puncture piston
  - Return Spring
  - Charge cups
  - Replacement O-rings
  - Powder measure vials
- Ejection Canisters (4 long and 4 standard length)
- Black Powder
- MJG igniters

4. Main Body Tube

- Main Parachute (Fruity Chute 84")
  - Parachute Protector
- 2 x quick links
- 30' Nylon Shock Chord
- 4 x 6-32 aluminum retention screws
- 4 x 4-40 shear pins
- SPOT3
- 57" fiberglass section

5. Motor

- CTI P98 – 6gXL Motor Casing
- Forward Closure
- Rear Closure
- Spacer
- Spanner Wrench

6. Documentation

- BigRedBee Manual
- TeleMega Manual
- Rocket Binder
- *RocketBoard* Binder

7. Flight Box

- Wire Stripper
- Extra Wire
- Wire Cutter
- Head lamp
  - Extra batteries
- Tweezers
- Large and small mixing sticks
- JB Weld 5-minute epoxy
- Aluminum Foil
- Sandpaper
- Nitrile Gloves
- Extra Shear Pins
- Extra retention screws
- Spare Batteries
- Screwdrivers
  - Hex
  - Standard
  - Phillips
  - Larger hex for switches
- Pencil
- Notepad
- Painter's tape
- Tube marking angle
- X-acto knife
- Multi-meter
- Shop towels
- Baby wipes
- 2 x GoPro session with charging cables
- Kenwood Radio
- Yagi Antenna
  - Large-to-small coax adapter
  - TeleDongle
- Super Lube
- High-vacuum grease
- Crescent wrench
- Dixie Cups
- Black Powder
- Extra igniters
- Dog Barf (Fire retardant stuffing)
- Personal laptop
  - Charger

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] P. Swintek, "Critical vulnerabilities in the Space Domain: using nanosatellites as an alternative to traditional satellite architectures," M.S. thesis, Dept. of Natl. Sec. Aff., NPS, Monterey, CA, USA, 2009.

[2] T. Yamagami, "Research on balloons to float over 50km altitude," JAXA, 2003. [Online]. Available: http://www.isas.jaxa.jp/e/special/2003/yamagami/index.shtml

[3] *NASA Sounding Rockets User Handbook*, NASA, Wallops Island, VA, USA, 2015. [Online]. Available: https://sites.wff.nasa.gov/code810/files/SRHB.pdf

[4] NASA Sounding Rocket Science, "NASA Sounding Rocket Program overview." Accessed March 14, 2019. [Online]. Available: https://rscience.gsfc.nasa.gov/srrov.html

[5] NASA Sounding Rockets Program Office, "Sounding rocket vehicles," Accessed March 14, 2019. [Online]. Available: https://sites.wff.nasa.gov/code810/vehicles.html

[6] NASA, "Sounding rockets annual report 2018," Wallops Island, VA, USA, 2018. [Online]. Available: https://sites.wff.nasa.gov/code810/files/Sounding_Rockets_Annual_Report_2018.pdf

[7] NASA, "National Aeronautics and Space Administration FY 2018 spending plan," July 2018. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/fy2018operatingplan_july2018.pdf

[8] H. Zell, "Opportunities for students," NASA, June 30, 2014. [Online]. Available: https://www.nasa.gov/centers/wallops/education/students/opportunities/index.html

[9] U.S. Air Force, "Experimental launch & test division/rocket system launch program," June 16, 2017. [Online]. Available: https://www.losangeles.af.mil/About-Us/Fact-Sheets/Article/1217574/experimental-launch-test-divisionrocket-system-launch-program/

[10] Launch Enterprise Directorate, "Small rocket program 4 (SRP-4) industry day," presented at Air Force Space Command, El Segundo, CA, USA, Mar. 1, 2016. [Online]. Available: https://www.fbo.gov/index?s=opportunity&mode=form&id=5349530e50c50edf64d3112080a3fc2a&tab=core&_cview=1

[11] Air Force Space Command Space and Missile Center Advanced Systems and Development Directorate Contraction Office, “SRP-3 ordering period justification and approval documentation,” Kirtland AFB, NM, USA, 2015. [Online]. Available: https://www.fbo.gov/index?s=opportunity&mode=form&id=05e04359882d0b1fd2ab3992ca14ce52&tab=core&_cview=0

[12] NAR, “High power rocketry.” Accessed March 16, 2019. [Online]. Available: https://www.nar.org/high-power-rocketry-info/

[13] National Fire Protection Agency, *NFPA 1127: Code for High Power Rocketry*. Quincy, Massachusetts, USA: NFPA, 2018. [Online]. Available: https://www.nfpa.org/codes-and-standards/all-codes-and-standards/list-of-codes-and-standards/detail?code=1127

[14] T. M. Sarradet, “A STEM based model rocketry curriculum: for the team America rocketry challenge,” M.A. thesis, Dept. of Graduate and Professional Studies, Cal. State Sac., Sacramento, CA, USA, 2009. [Online]. Available: https://www.nar.org/pdf/STEM-TARC%20Model%20Rocketry%20Curriculum%20%28Sarradet%29.pdf

[15] O. Yakimenko, NPS SE and MAE Department professor, interview, Oct. 2018.

[16] Colorado Space News, “CSXT GO FAST! Rocket confirms multiple world records.” Accessed March 17, 2019. [Online]. Available: http://www.coloradospacenews.com/csxt-go-fast-rocket-confirms-multiple-world-records/

[17] BPS.Space, “About.” Accessed March 18, 2019. [Online]. Available: https://bps.space/about

[18] Madcow Rocketry, “Kits – all.” Accessed March 18, 2019. [Online]. Available: https://www.madcowrocketry.com/kits/

[19] Animal Motor Works, “CTI.” Accessed March 18, 2019. [Online]. Available: http://cart.amwprox.com/index.php?option=com_virtuemart&view=category&virtuemart_category_id=3&Itemid=470

[20] R. DeHate, Owner of Animal Motor Works, email, Oct. 2018.

[21] Federal Aviation Administration, “Code of Federal Regulations Title 14 Part 101,” 1 January 2019. [Online]. Available: https://www.ecfr.gov/cgi-bin/text-idx?SID=f7bff3c340f72d43126e111c1f51870e&mc=true&tpl=/ecfrbrowse/Title14/14cfr101_main_02.tpl

[22] Federal Aviation Administration, “Code of Federal Regulations Title 14 §101.22,” 1 January 2019. [Online]. Available: https://www.ecfr.gov/cgi-bin/text-idx?SID=f7bff3c340f72d43126e111c1f51870e&mc=true&node=se14.2.101_122&rgn=div8

[23] T. J. Pierce, Professional Engineer in the discipline of Fire Protection Engineering, interview, Mar. 2019.

[24] Tripoli Rocket Association, “Certification.” Accessed March 16, 2019. [Online]. Available: http://www.tripoli.org/Certification

[25] National Association of Rocketry, “Find a local NAR club (section).” Accessed March 16, 2019. [Online]. Available: https://www.nar.org/find-a-local-club/

[26] *DoD Ammunition and Explosives Safety Standards: General Explosives Safety Information and Requirements*, DoD Manual 6055.09-M, Under Secretary of Defense (AT&L), Washington, DC, USA, 2012. [Online]. Available: https://wbdg.org/ffc/dod/manuals/dod-605509-m-volume-1

[27] Naval Sea Systems Command, “Naval Ordnance Safety and Security Activity (NOSSA).” Accessed: March 17, 2019. [Online]. Available: https://www.navsea.navy.mil/Home/NOSSA.aspx

[28] *Naval Sea Systems Command Ordnance Pamphlet 5*, NAVSEA OP 5, v.1, rev. 7, Director of Department of Navy Weapons, Ordnance and Explosives Safety, Washington Navy Yard, DC, USA, 2017.

[29] *Naval Personnel Ammunition and Explosive Handing Qualification and Certification Program*, OPNAVINST 8023.24C, Department of the Navy, Washington, DC, USA, 2014. [Online]. Available: http://navybmr.com/study%20material/OPNAVINST%208023.24C.pdf

[30] S. Schnur, NPS Explosive Safety Officer and QUAL/CERT Board Chair, interview, Sep. 2018.

[31] White Sands Missile Range, “Missiles/Rockets,” October 26, 2018. [Online]. Available: https://www.wsmr.army.mil/testcenter/services/wpt/Pages/Missiles.aspx

[32] S. Bogle, White Sands Missile Range business analyst, email, Jan. 2019.

[33] Commander, Navy Installations Command, “Naval Air Weapons Station China Lake.” Accessed April 2, 2019. [Online]. Available: https://www.cnic.navy.mil/regions/cnrsw/installations/naws_china_lake.html

[34] Camp Roberts Ammunition Supply Point Chief, interview, Oct. 2018.

[35] R. Wuerz, White Sands Missile Range business analyst, interview, Apr. 2019.

[36] Friends of Amateur Rocketry, "Friends of Amateur Rocketry." Accessed March 17, 2019. [Online]. Available: https://friendsofamateurrocketry.org/

[37] Friends of Amateur Rocketry, "Licenses." Accessed March 17, 2019. [Online]. Available: https://friendsofamateurrocketry.org/licenses/

[38] Google. "Friends of Amateur Rocketry." [Online]. Available: https://www.google.com/maps/place/Friends+of+Amateur+Rocketry/@35.3467799,-117.810393,17z/data=!3m1!4b1!4m5!3m4!1s0x80c17c12b6374b4b:0x76a569600206fb35!8m2!3d35.3467755!4d-117.8082043

[39] U.S. Geological Survey. "Friends of Amateur Rocketry operating area." [Online]. Available: https://viewer.nationalmap.gov/basic/?basemap=b1&category=histtopo,ustopo&title=Map%20View#productSearch

[40] Federal Aviation Administration, "Code of Federal Regulations Title 14 §101.25," 1 January 2019. [Online]. Available: https://www.ecfr.gov/cgi-bin/retrieveECFR?gp=&SID=ac3463b7201f5e3690ae6d0a214ad1cd&mc=true&n=sp14.2.101.c&r=SUBPART&ty=HTML#se14.2.101_125

[41] SkyVector. "FAR rocket range." [Online]. Available: https://skyvector.com/

[42] Friends of Amateur Rocketry, "Mission." Accessed April 2, 2019. [Online]. Available: https://friendsofamateurrocketry.org/mission/

[43] Friends of Amateur Rocketry, "Calendar." Accessed April 2, 2019. [Online]. https://calendar.google.com/calendar/embed?src=db1o3d1in7nsnoeo1ud4bebprg%40group.calendar.google.com&ctz=America%2FLos_Angeles

[44] Friends of Amateur Rocketry, "Fees." Accessed April 2, 2019. [Online]. Available: https://friendsofamateurrocketry.org/fees/

[45] J. Coker, "Simulation data – AeroTech H128," ThrustCurve, April 4, 2019. [Online]. Available: http://www.thrustcurve.org/simfilesearch.jsp?id=919

[46] M. Canepa, *Modern High-Power Rocketry 2*. Victoria, British Columbia, Canada: Trafford, 2005.

[47] T. Van Milligan, "Pressure relief holes," *Peak of Flight Newsletter*, Nov. 19 2001. [Online]. Available: https://www.apogeerockets.com/education/downloads/Newsletter68.pdf

[48] J. Coker, "Simulation data – AeroTech J90," ThrustCurve, April 8, 2019. [Online]. Available: http://www.thrustcurve.org/simfilesearch.jsp?id=944

[49] Duracell, “Duracell technical library.” Accessed July 30, 2018. [Online]. Available: https://www.duracell.com/en-ca/techlibrary/product-technical-data-sheets

[50] J. Coker, “Simulation data – AeroTech M650,” ThrustCurve, April 8, 2019. [Online]. Available: http://www.thrustcurve.org/simfilesearch.jsp?id=980

[51] D. Pierce, “Level 3 certification attempt documentation,” unpublished. Available: ssagcommon$\ High Power Rocket\Reports\Level 3 Documentation

[52] J. Coker, “Simulation data – AeroTech M1315,” ThrustCurve, April 9, 2019. [Online]. Available: http://www.thrustcurve.org/simfilesearch.jsp?id=983

[54] J. Schiff and M. Aspell, “How to achieve extreme altitude deployment,” *Peak of Flight*, May 10, 2011. [Online]. Available: https://www.apogeerockets.com/education/downloads/Newsletter286.pdf

[55] B. Bollermann et al., “Design, development and flight test of the super loki dart meteorological rocket systems,” Springfield, VA, USA, 2015. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/750796.pdf

[56] M. Wade, “Loki chronology,” Astronautix. [Online]. Available: https://web.archive.org/web/20100102233831/http://astronautix.com/lvs/loki.htm#chrono

[57] NASA, “Rocket stability,” June 12, 2014. [Online]. Available: https://www.grc.nasa.gov/www/k-12/rocket/rktstab.html

[58] J. S. Barrowman, “The practical calculation of the aerodynamic characteristics of slender finned vehicles,” M.S. thesis, Dept. of Eng. and Arch., Cath. Univ. of Amer., Washington, DC, USA, 1967. [Online]. Available: http://mae-nas.eng.usu.edu/MAE_5900_Web/5900/USLI_2010/Flight_Mechanics/Barrowman.pdf

[59] E. W. Perkins, L. H. Jorgensen, S. C. Sommer, “Investigation of the drag of various axially symmetric nose shapes of finess ratio 3 for mach numbers 1.24 to 7.4,” NACA, Mountain View, CA, USA, Rep. 1386. [Online]. Available: https://digital.library.unt.edu/ark:/67531/metadc65615/m1/1/

[60] Fruity Chutes, “Home.” Accessed April 11, 2019. [Online]. Available: https://fruitychutes.com/

[61] Dragon Plate, “Carbon fiber birch core.” Accessed April 11, 2019. [Online]. Available: https://dragonplate.com/Carbon-Fiber-Birch-Core

[62] Tinder Rocketry, “A discussion on CO2 ejection.” Accessed April 11, 2019. [Online]. Available: https://www.tinderrocketry.com/rocketry-co2-ejection-system

[63] Skylighter, “Black powder grades, sizes and mesh.” Accessed April 11, 2019. [Online]. Available: https://www.skylighter.com/blogs/fireworks-information/black-powder-grades-sizes-mesh

[64] R. Sasse. “A comprehensive review of black powder,” U.S. Army Ballistic Research Lab., Aberdeen Proving Ground, MD, USA, BRL-TR-2630, 1985. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a150455.pdf

[65] Feretich, “Shear pin/screw calculations.” Accessed April 11, 2019. [Online]. Available: http://www.feretich.com/Rocketry/Resources/shearPins.html

[66] *Tinder Rocketry’s –RAPTOR CO2 Ejection System Instruction Manual,* Tinder Rocketry. [Online]. Available: https://docs.wixstatic.com/ugd/b73de9_1cf9c63dfeaa4018908df6f9ba02c0ca.pdf

[67] Altus Metrum, “TeleMega,” April 22, 2017. [Online]. Available: https://altusmetrum.org/TeleMega/

[68] Altus Metrum, “TeleDongle,” April 22, 2017. [Online]. Available: https://altusmetrum.org/ TeleDongle/

[69] Altus Metrum, “AltOS,” August 5, 2018. [Online]. Available: https://altusmetrum.org/ AltOS

[70] BigRedBee, “70cm 100mw GPS/APRS transmitter,” Accessed April 11, 2019. [Online]. Available: https://shop.bigredbee.com/products/70cm-100mw-gps-aprs-transmitter

[71] *TH-72A/ TH-72E Instruction Manual*, JVC Kenwood Corp, Yokohama, Kanagawa Prefecture, Japan.

[72] STMicroelectronics, *LSM9DS1 iNEMO inertial module*, 025715 Rev. 3, 2015. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/1e/3f/2a/d6/25/eb/48/46/DM00103319.pdf/files/DM00103319.pdf/jcr:content/translations/en.DM00103319.pdf

[73] Analog Devices, *ADXL377 Accelerometer,* Rev. A, 2018. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL377.pdf

[74] NXP, *MPL3115A2 Pressure Sensor*, Rev. 8, 2018. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MPL3115A2.pdf

[75] Microhard, *n920x2 900 MHz Wireless Modem*. [Online]. Available: http://www.microhardcorp.com/brochures/n920X2.Brochure.Rev.1.0.pdf

[76] ON Semiconductor, *CAT24C32 EEPROM,* CAT24C32/D, Rev. 26, 2018. [Online]. Available: https://www.onsemi.cn/PowerSolutions/document/CAT24C32-D.PDF

[77] J. Pross, "Rocket-delivered short-term expeditionary beyond line-of-sight narrowband communications relay," unpublished.

[53] Friends of Amateur Rocketry, "Launchers." Accessed April 11, 2019. [Online]. Available: https://friendsofamateurrocketry.org/launchers/

[78] G. Minelli, private communication, May 2019

[79] Federal Aviation Administration, "Code of Federal Regulations Title 14 §101.29," 1 January 2019. [Online]. Available: https://www.ecfr.gov/cgi-bin/retrieveECFR?gp=&SID=e3f6200e7ca33fa07d84ce90964cc98d&mc=true&n=sp14.2.101.c&r=SUBPART&ty=HTML#se14.2.101_129

[80] M. Williams and P. Peck, 2018. IMU.py [Online]. Available: https://github.com/ozzmaker/BerryIMU/blob/master/python-BerryIMU-measure-G/IMU.py

[81] M. Williams and P. Peck, 2018. LSM9DS1.py, [Online]. Available: https://github.com/ozzmaker/BerryIMU/blob/master/python-BerryIMU-measure-G/LSM9DS1.py

[82] A. Tridgell, 2012. ublox.py, [Online]. Available: https://github.com/emlid/Navio2/blob/master/Python/navio/ublox.py

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California